Security – Secure Socket Layer, Authentication

Distributed Systems and Security

Lecture 21

SSL - Secure Socket Layer

Now known as

TLS – Transport Layer Security

Application – Web Security

How can we authenticate the server?

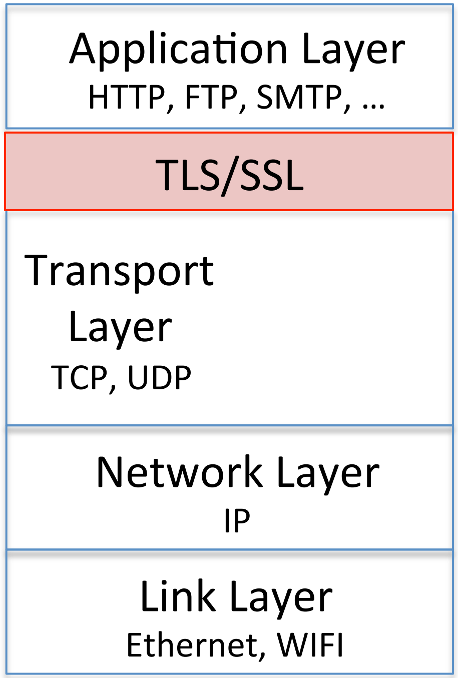
How can we establish a secure communication channel?

Web Server

- Secure communication between a web client and a server
- SSL Secure Socket Layer / TLS Transport Layer Security
  - Uses symmetric key for encryption/decryption of transferred data
  - Uses public key cryptography for secure exchange of a secret symmetric key and authentication of servers (and clients) with the exchange of certificates
  - Allows the use of a variety of ciphers and hash functions

SSL Secure Socket Layer

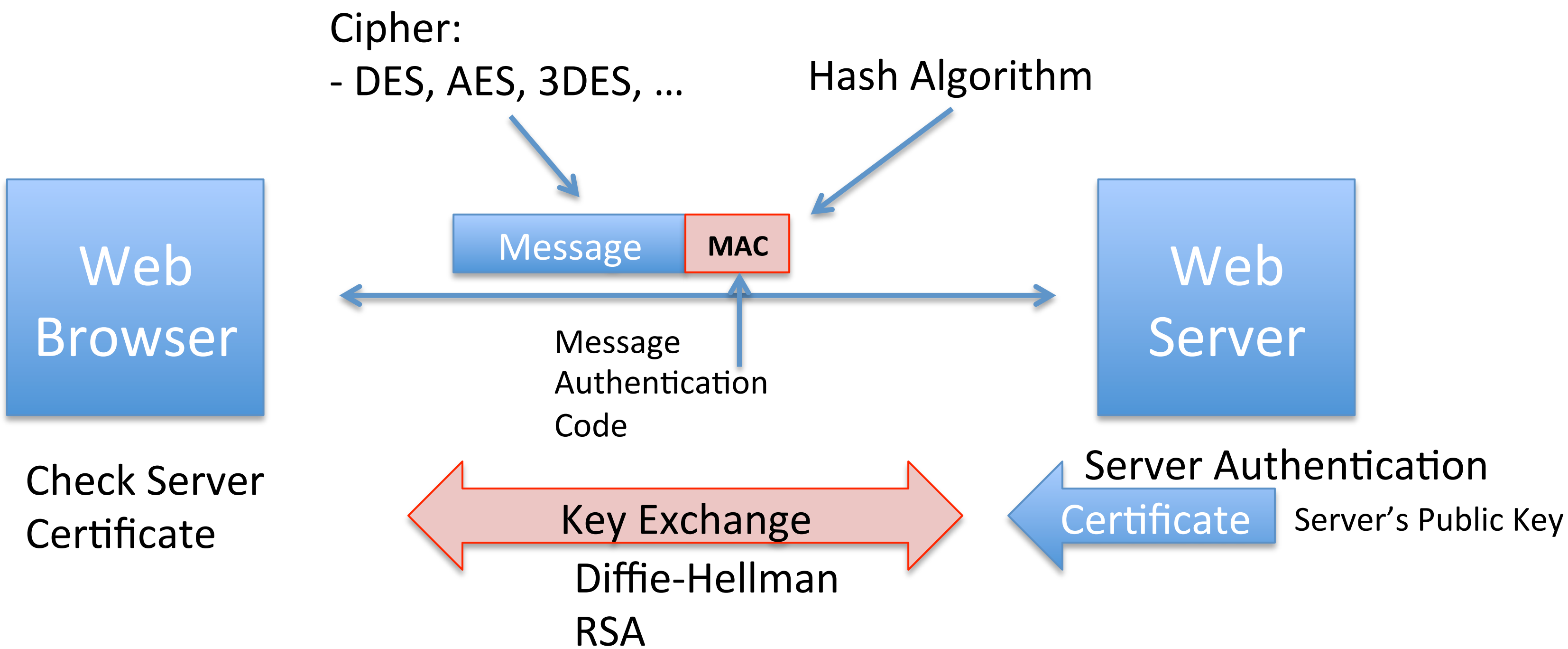
- Developed by Netscape in 1995 for secure and authenticated connections between web browsers and servers
- SSL provides transport layer security
- Transport Layer Security TLS
  - TLS 1.0 succeeded SSL 3.0
- TLS / SSL essential for security and privacy on the web
- Builds directly on top of the Transport Layer Protocol TCP, provides secure communication for the Transport Layer (via TCP/IP)
  - Layer 3: IP (Internet Protocol), contains addressing and (limited) control information
  - Layer 4: TCP (Transport Control Protocol), provides end-to-end connection-oriented packet delivery
- Should be "transparent" to applications, allows them to communicate with servers in a secure fashion



SSL Secure Socket Layer

- SSL can be viewed as a security layer that sits between the application layer and the transport layer
  - The client hands over data to SSL (e.g. An HTTP message for a web server), SSL then encrypts this data and writes it to a TCP socket
  - The server receives this encrypted data via its TCP socket, SSL takes this data, decrypts it and directs this data to the server for processing
- SSL uses symmetric key cryptography for encryption and decryption of data that is transferred
- To Do:
  - Verify that server is trustworthy (certificates)
  - Exchange a symmetric key between server and client

SSL Secure Socket Layer



- Before encrypted communication
  - Negotiate cipher suite: crypto algorithm, hash algorithms for MAC
  - Authenticate Server (one-way authentication)
  - (maybe server also requests client authentication with certificate: two-way authentication)
  - Exchange information for secret key with Diffie-Hellman, RSA etc. key exchange

Key Exchange

- Diffie-Hellman:
  - Public agreement: a large prime number **p** and a small integer value **g (g < p)**
  - Private: each communication partner chooses a large random secret value: **a < p-1**, **b < p-1**
  - Public exchange: values A and B
    - $A = g^a \mod p$
    - $B = g^b \mod p$
  - Calculation of secret symmetric key at each site
- RSA
  - RSA calculation results in a public key {n,e} and a private key d
  - Server sends certificate (signed by a CA), which contains the server's public key
    - If only server sends a certificate, then this is a one-way authentication
  - Client sends a random secret key information encrypted with the server's public key

SSL/TLS Transport Layer Security

- TLS is the successor of SSL
  - SSL 3.0 is succeeded by TLS (in the TLS handshake regarded as SSL 3.1)
  - SSL often used synonymous for both TLS and SSL
- Objectives
  - Provide privacy and data integrity between a client and server
- Privacy
  - Symmetric key cryptography is used for data transmission
- Authentication
  - Public key cryptography is used to authenticate communication partners
    - Although an optional step, usually the server transmits a certificate signed by a CA
- Message integrity
  - For each message transmitted, a Message Authentication Code (MAC) is generated, which is a hash (message digest) of the message
  - Prevents undetected loss or alteration of data during transmission (man-in-the-middle attack)
- Forward Secrecy
  - Any future disclosure of secret keys cannot be used to decrypt TLS communication recorded in the past

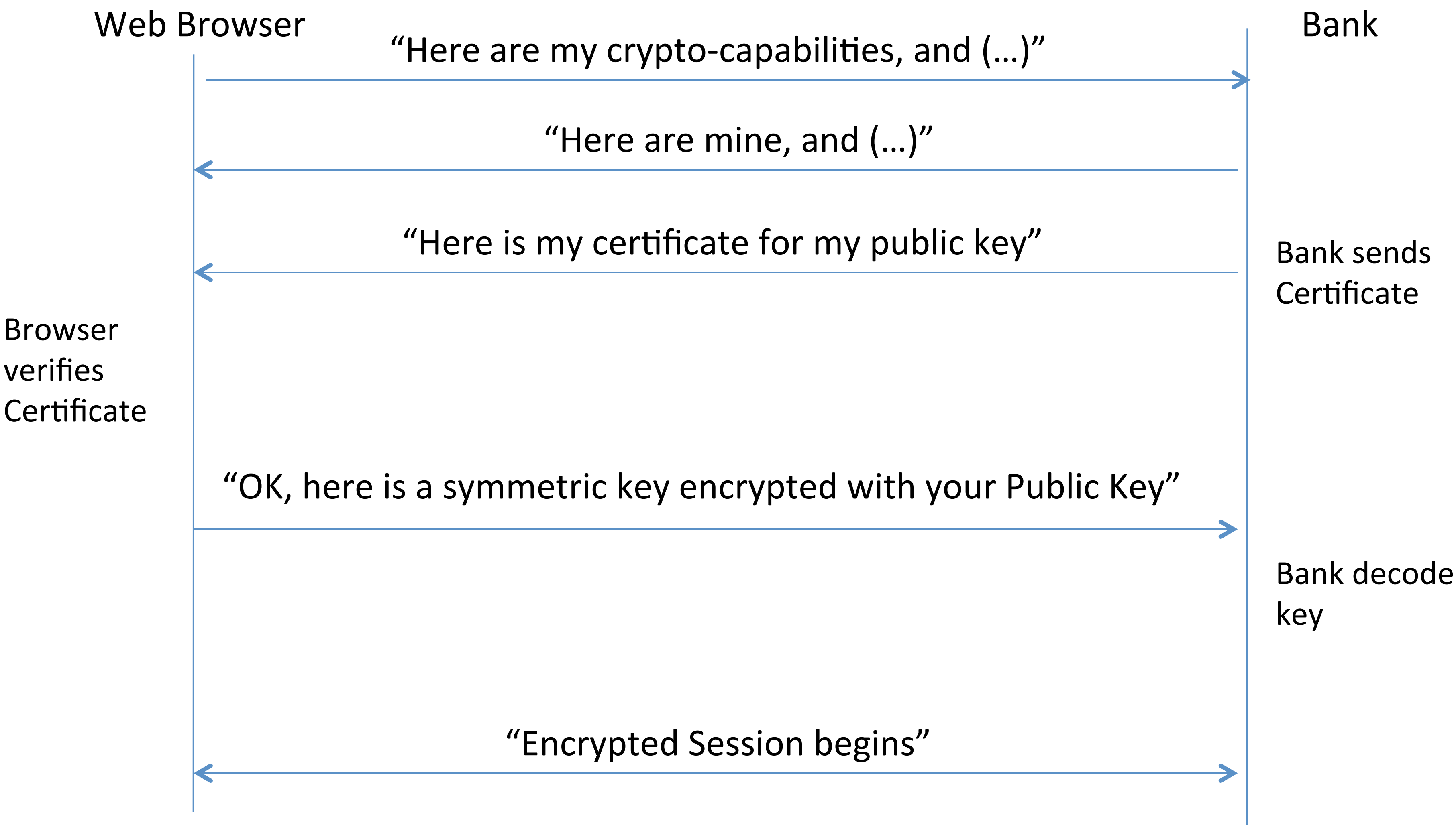
Objective and Properties

- Safeguard privacy and data integrity between two computer applications
- Properties of Client – Server communication secured by TLS
  - Private connection:
    - Symmetric cryptography used to encrypt data transmitted
  - Handshake Protocol:
    - Secret negotiation at start of session
      - Which encryption algorithm to be used
      - Keys for symmetric encryption exchanged
    - Negotiation is secure: negotiated secret unavailable to eavesdropper
    - Negotiation is reliable: attacker cannot modify the communication without being detected
  - Authentication:
    - Authentication of parties via public key cryptography
  - Reliable connection
    - Messages include a MAC (message authentication code) to prevent undetected loss or alteration of the data during transmission
- Additional privacy-related properties
  - Forward secrecy, when ephemeral Diffie-Hellman key exchange is used (not fixed, but temporary public keys)

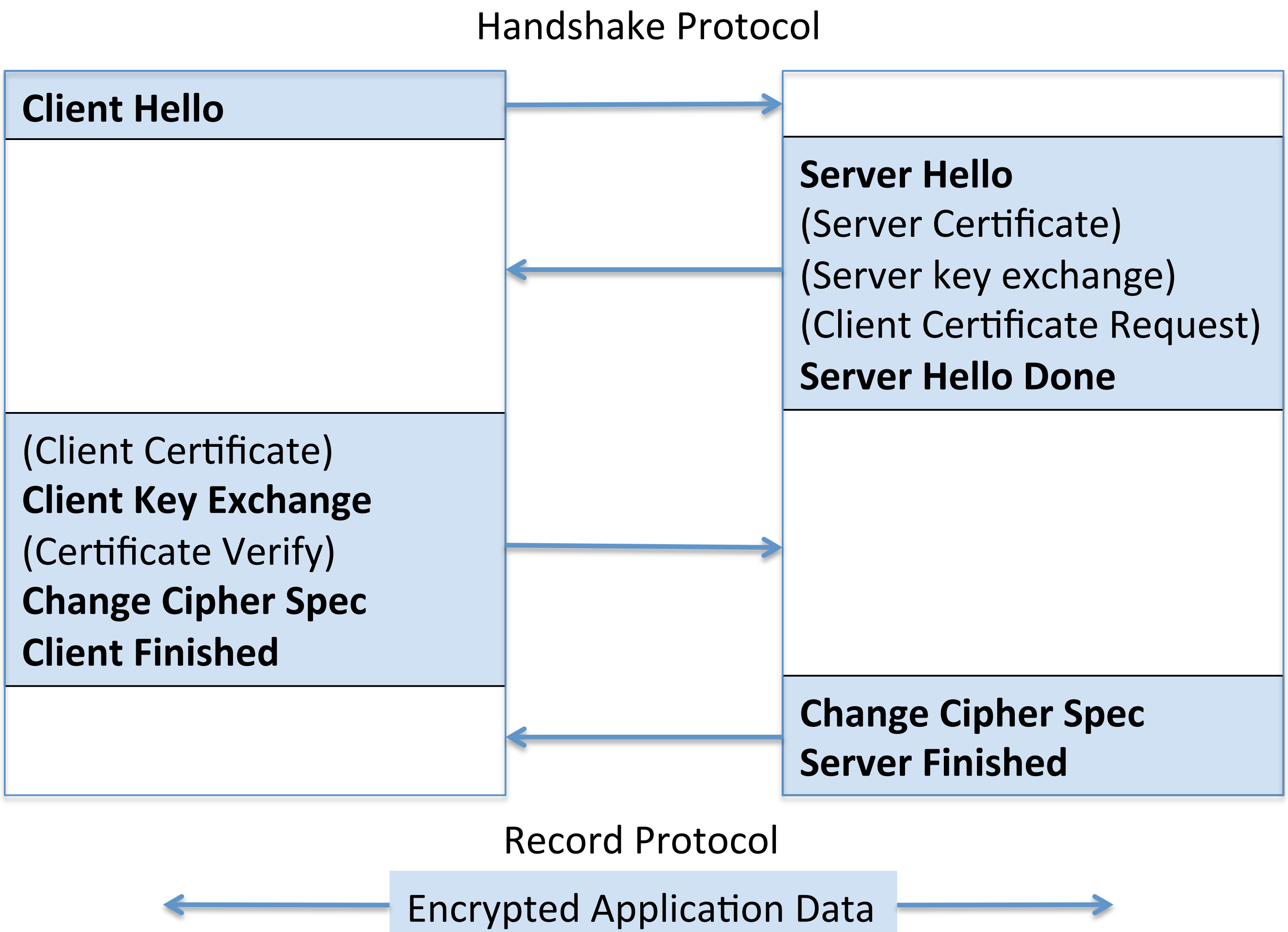
SSL/TLS Protocols

- ***Handshake Protocol***
  - Negotiation phase to establish a secure TLS session between browser and web server
    - Negotiate a Cipher Suite: encryption algorithms, hash, secret key
    - Negotiation is secure: negotiated secret unavailable to eavesdropper
    - Negotiation is reliable: attacker cannot modify the communication without being detected
  - TLS uses public key cryptography for establishing a secure TLS session
    - Authentication of web server and client
    - exchange of a symmetric key for communication
- ***Record Protocol***
  - Communication phase
    - TLS uses symmetric key cryptography for transmission of data during a secure TLS session between a browser and a web server
  - Use of a hash function to generate the MAC Message Authentication Code as a Message Digest
    - Message is encrypted together with the MAC

SSL Handshake
One-way Authentication, RSA



SSL/TLS Handshake



SSL Handshake Phases

1. Browser sends SSL version and cryptographic preferences (e.g. RSA for key exchange, DES for communication) to server
2. Server sends SSL version, cryptographic preferences, and certificate; certificate (certified by some CA) contains server's encrypted RSA Public Key
3. Browser lookup of certificates CA; if CA is in the browsers list of trusted CA's, the CA's Public Key is used to (a) validate the certificate and (b) decrypt the server's RSA Public Key
4. Browser generates a symmetric secret (private) key (called the "session key"), encodes it with the server's RSA Public Key and sends it to the server
5. Server uses its RSA private key to decrypt the received session key for this SSL communication
6. Handshake finished, browser and server start communicating using the secret session key

1-way (server) authentication and RSA key exchange.

Cipher Suite Negotiation

- Client and Server choose a cipher suite (SSL 3.0 defines 31 cipher suites)
- A Cipher Suite is defined by the following components
  - Key exchange Method
    - SSL 2.0 supports only RSA key exchange
    - TLS supports:
      - RSA key exchange when certificates are used
      - Diffie-Hellman key exchange when there has been no prior communication between client and server
  - Cipher for Data Transfer
    - TLS / SSL uses symmetric key cryptography, various choices of encryption algorithms:
      - DES, 3DES, IDEA, etc.
  - Method for creating the Message Authentication Code (MAC) / Message Digest Function choice
    - No Digest
    - MD5
    - Secure Hash Algorithm (SHA-1)
    - HMAC

Client Hello Message

- Client initiates session by sending a “Client Hello” message
  - Version of SSL protocol:
    - Client sends version number of highest SSL/TLS protocol it supports (Version 2 – SSL 2.0, Version 3 – SSL 3.0, Version 3.1 – TLS)
  - ClientRandom [32bit]:
    - Client generates a 4 byte random number, consists of
      - Client’s date and time, 28 bit randomly generated number
    - (will be ultimately used, together with a corresponding server random number, to generate a master secret for deriving the encryption keys for the session)
  - Session ID (empty, when new session):
    - Can be used to resume a session, without repeating the handshake protocol (avoiding time consuming public key encryption operations)
  - Cipher Suite
    - Transmits a list of cipher suites available to the client
    - E.g.: “TLS_RSA_WITH_DES_CBC_SHA”
      - TLS protocol, RSA key exchange algorithm, DES_CBC encryption, SHA hash function
  - Compression Algorithm
    - Messages may be compressed before encryption (not recommended)

https://technet.microsoft.com/en-us/library/cc785811(WS.10).aspx

Client Hello Message, Example

```
ClientVersion 3,1
ClientRandom[32]
SessionID: None (new session)
Suggested Cipher Suites:
  TLS_RSA_WITH_3DES_EDE_CBC_SHA
  TLS_RSA_WITH_DES_CBC_SHA
Suggested Compression Algorithm: NONE
```

- Example “Client Hello Message”

https://technet.microsoft.com/en-us/library/cc785811(WS.10).aspx

Server Hello Message

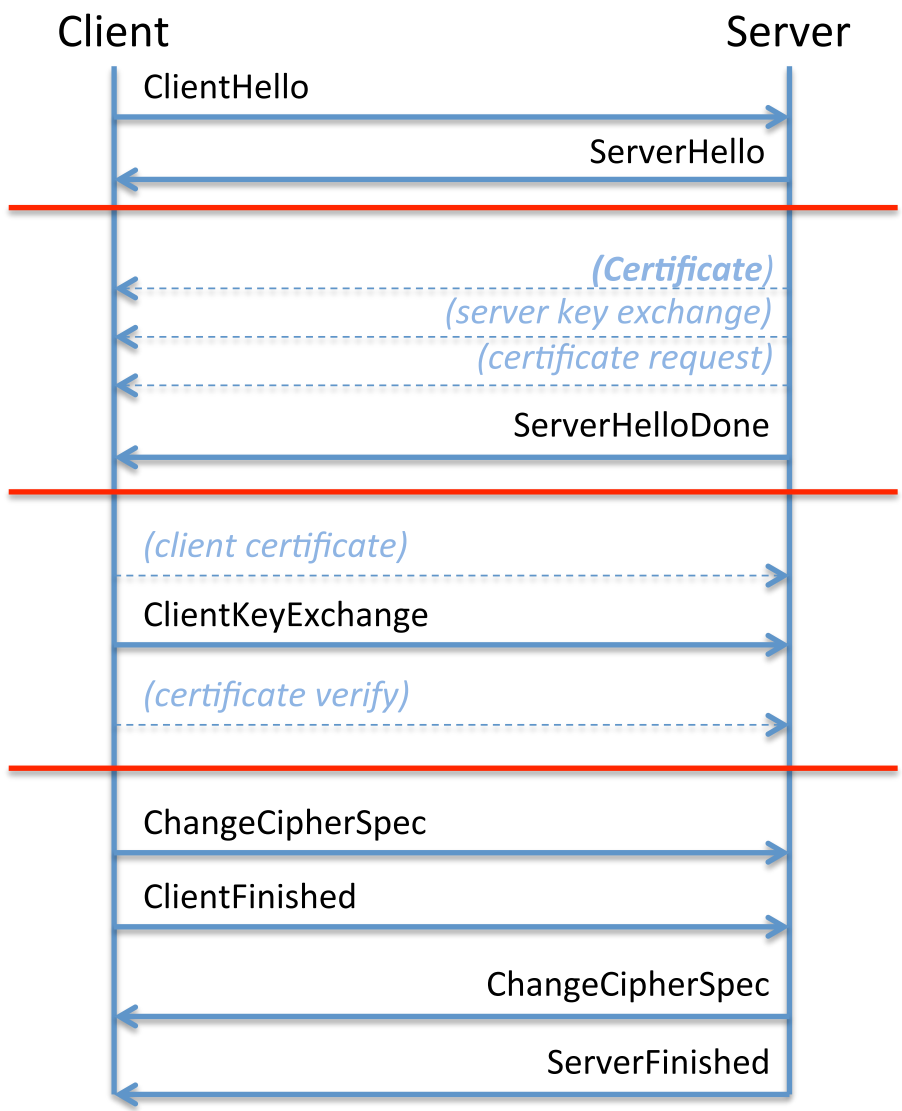
- Server responds to Client Hello message:
  - Version of SSL protocol
    - Selects highest possible SSL protocol according to client's specification
  - ServerRandom [32bit]:
    - Server also generates a 4-bit random number, consists of:
      - Server's date and time, 28-bit randomly generated number
      - (ClientRandom and ServerRandom will be used for generating a master secret for deriving the encryption keys)
  - Session ID:
    - Server generates a new session ID
      - if (a) client send empty ID, or (b) server refuses to resume session despite client request
    - Server resumes session with session ID sent by client
    - Session ID is not generated – this is a one-off session, which cannot be resumed
  - Cipher Suite
    - Server selects strongest cipher suite supported by both parties
  - Compression algorithm

Server Hello Message, Example

Version 3,1
ServerRandom[32]
SessionID: bd608869f0c629767ea7e3ebf7a63bdcffb0ef58b1b941e6b0c044acb6820a77
Use Cipher Suite:
TLS_RSA_WITH_3DES_EDE_CBC_SHA
Compression Algorithm: NONE

- Example content of the Server Hello Message

TLS Handshake



Phase 1

Establish security capabilities:
- Protocol version, session ID,
  cipher suite,
  random numbers for key exchange

Phase 2

Server may send (optional):
- certificate,
- Public info for key exchange,
- Request for client certificate

Server signals end of hello message phase

Phase 3

Client may send (optional):
- certificate, certificate verification

Client sends
- Public info for key exchange,

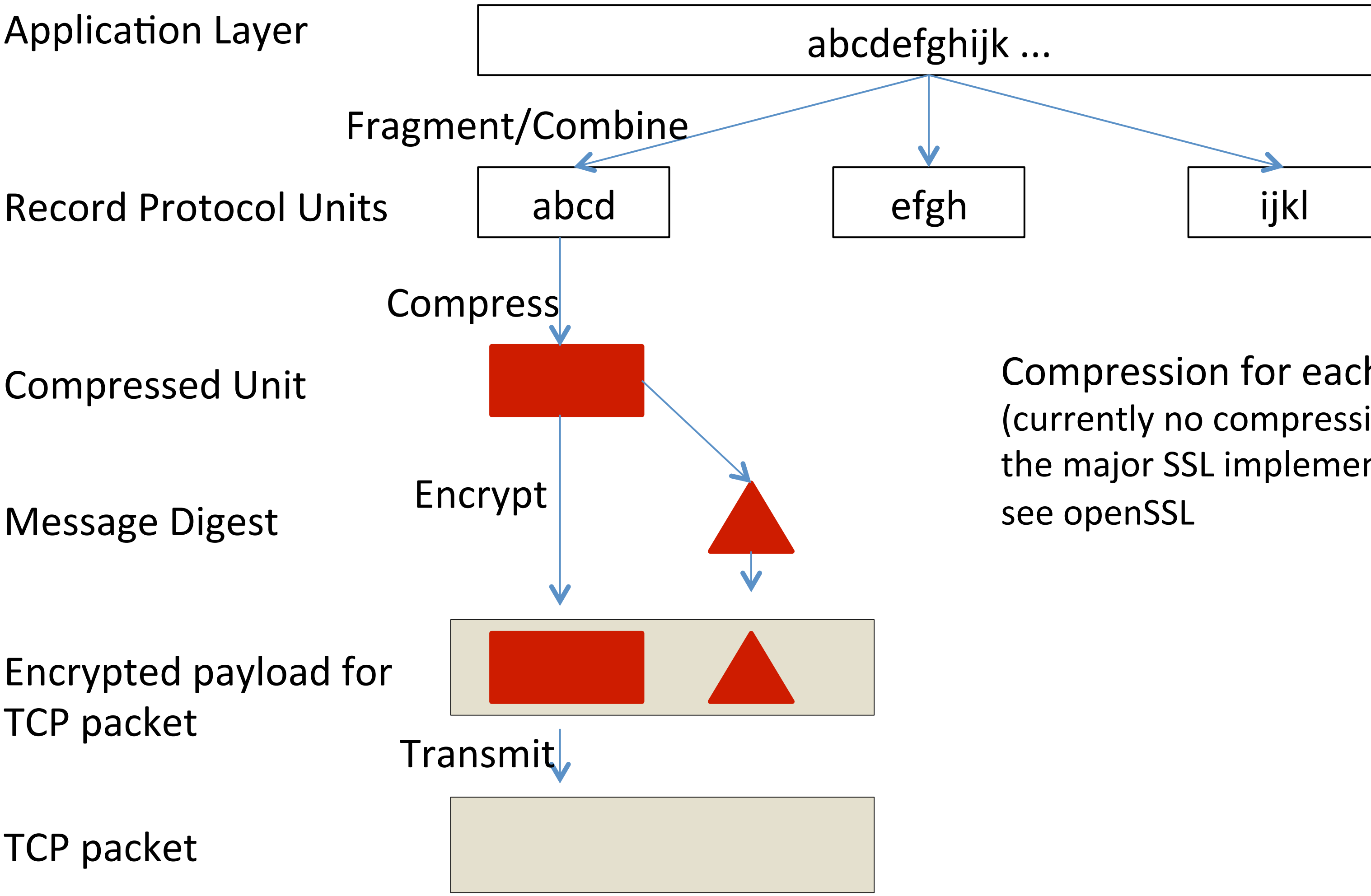
Phase 4

Client and Server
- Change cipher suite
- Finish handshake protocol

SSL: Hash Function for MAC

- Hash function, as agreed during handshake, is used to create a Message Authentication Code (MAC)
  - Is a "Message Digest"
  - Message Digest Function is a secure one-way hash function
    - The outcome of applying a message digest function, such as MD5, is a Message Digest
- Hash Functions
  - MD5, 128-bit hash, creates a 128-bit Digest
  - Secure Hash Algorithm SHA-1, 160-bit hash, creates a 160-bit Digest
- Message and MAC are encrypted together with session key to saveguard and prove integrity of transferred data
- A MAC is like a digital signature, however, symmetric keys are used

Data Transfer
SSL Record Protocol



Compression for each unit (currently no compression in most of the major SSL implementations), but see openSSL

Security - Authentication

Authentication

- Verification of identity:
  - Authentication is the act of verifying someone's identity
  - Users have to follow a particular procedure to prove their identity to a service
- It doesn't specify what a user is allowed or not allowed to do (that would be authorisation)

Authentication

- *Authentication* - verifying who you are
- *Authorisation* - verifying what you are allowed to do (what services to use, what actions to perform), after a server has authenticated you
- Access Control is a mechanism to control (allow / deny) access to resources
  - Examples include files, services, machines on a network etc.

Authentication

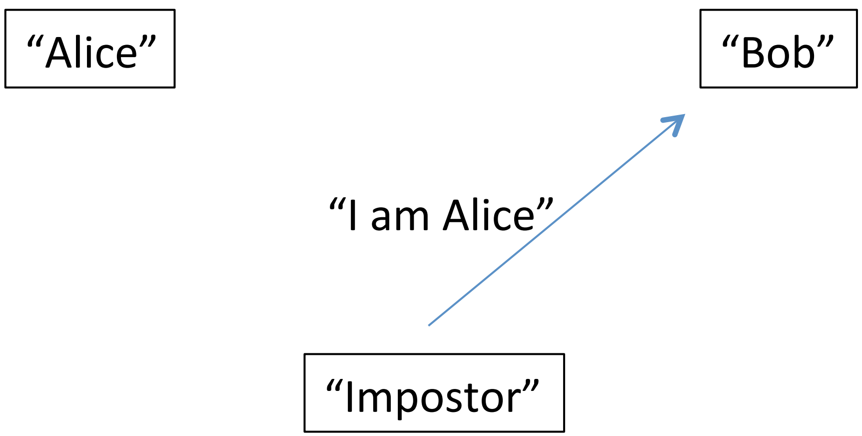
- Goal: Users want to prove their identity to a server

“Alice” $\xrightarrow{\text{“I am Alice”}}$ “Bob”

In a network, Bob cannot “see” Alice, so how can he be sure about the identity / origin of the received request?

Authentication – Imposter Problem

- Goal: Users want to prove their identity to a server

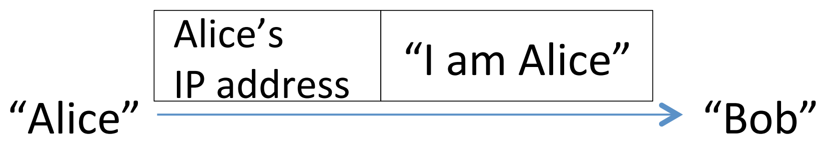


In a network, Bob cannot “see” Alice

An impostor can declare herself to be “Alice”

Authentication with IP Address

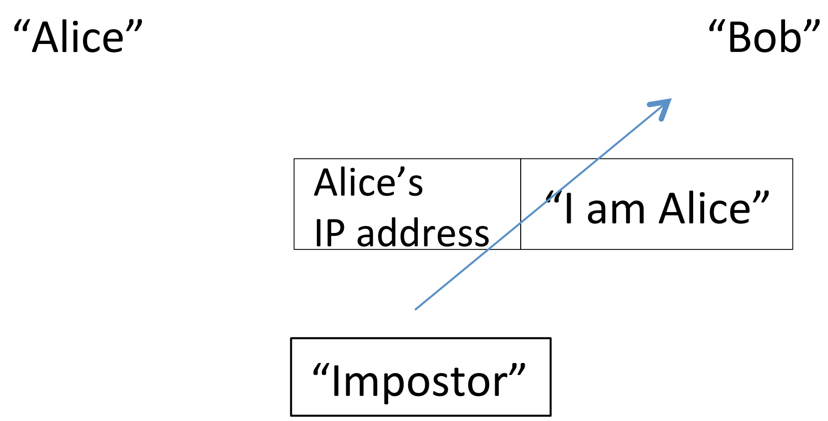
- Goal: Users want to prove their identity to a server



Failure scenario?

Authentication – Spoofing Problem

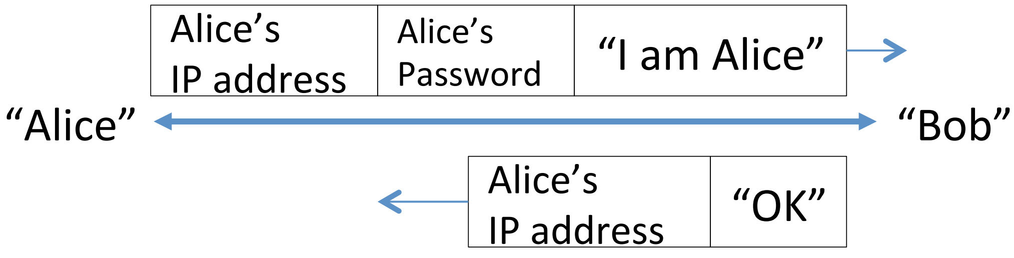
- Goal: Users want to prove their identity to a server



Failure scenario: Impostor can create a packet “spoofing” Alice’s IP address

Authentication with Password

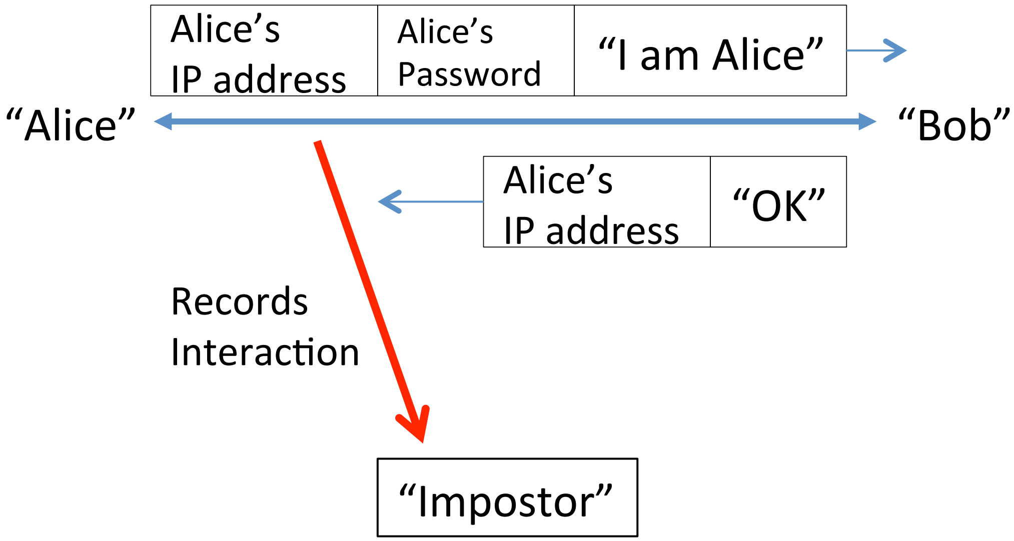
- Use Password: transmit password in the clear over network
  - Main Problem: eavesdropping / interception



Failure scenario?

Authentication – Playback Attack

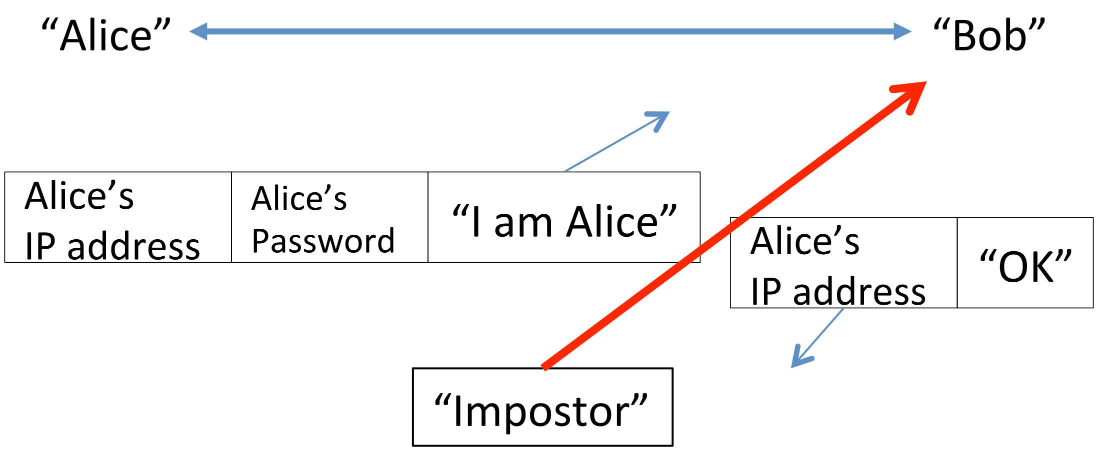
- Use Password: transmit password in the clear over network
  - "Playback Attack"



Playback Attack: Impostor records the IP packets exchanged

Authentication – Playback Attack

- Use Password: transmit password in the clear over network
  - "Playback Attack"



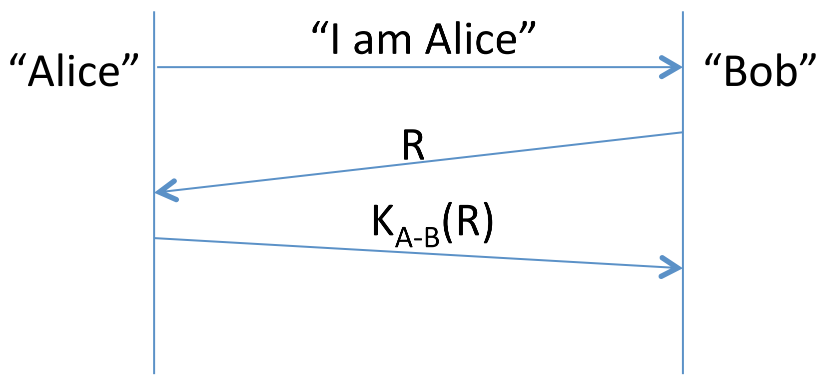
Playback Attack: Impostor records the IP packets exchanged and plays it back to receiver

Cryptographic Authentication

- No password is transmitted over the network
  - Users prove their identity to a service by performing a cryptographic operation
- Cryptographic operations are based on the user's secret key

Authentication with Symmetric Keys

- Goal: Avoid playback attacks by using a nonce
  - Use a “nonce” (a number used only once), issued by the receiver
  - Ensures that old communications cannot be reused in a replay attack
- Scenario:
  - Alice sends identification
  - **To test/prove that Alice is the sender, Bob sends the nonce “R” back**
  - Alice must return R, encrypted with the symmetric key $K_{A\text{-}B}$
  - Bob can verify Alice by decrypting the nonce R with the symmetric key $K_{A\text{-}B}$



Only “Alice” should know key $K_{A\text{-}B}$ to encrypt the nonce.
Failures, drawbacks?

Authentication with Symmetric Keys

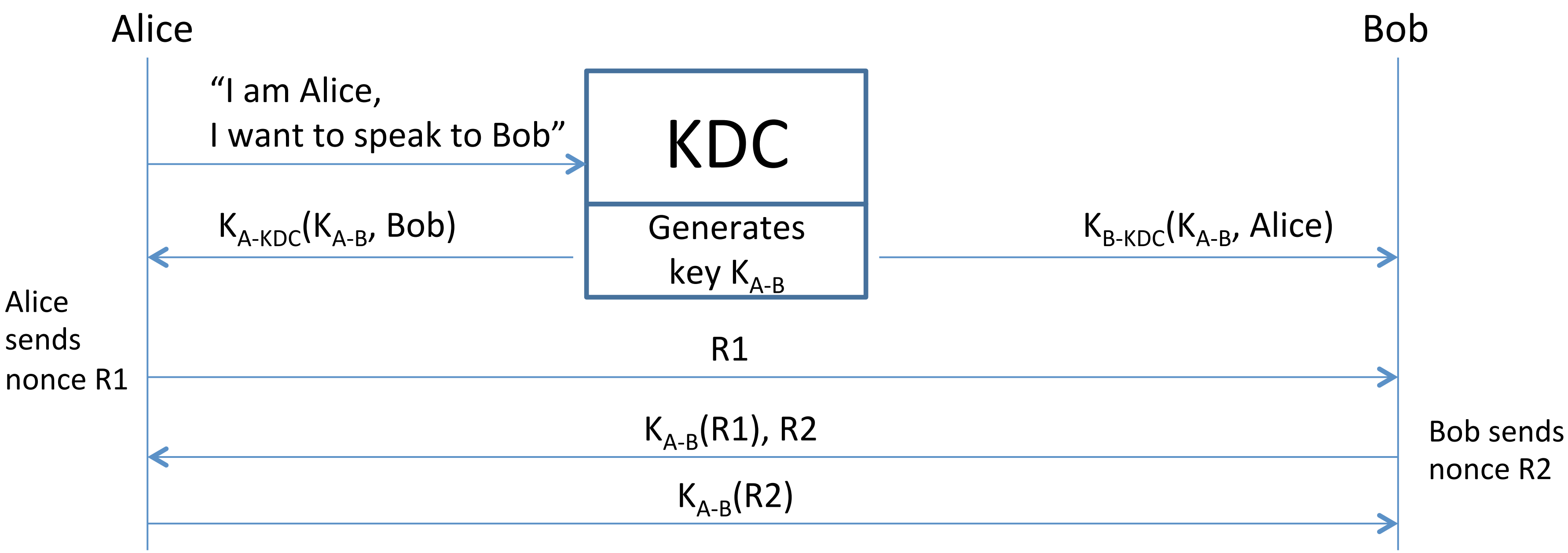
- Problems
  - Poor scaling properties
    - Generalising the model for **m users and n services** requires that m x n secret keys are distributed beforehand
- Possible Improvement
  - Use a trusted 3rd party, with which each user and service shares a secret key
    - These are then "only" m + n secret keys to be distributed

Key Distribution Center (KDC)

- **A Key Distribution Center** is a *trusted third party* that shares different secret keys with each registered user
- The KDC introduces "*Mediated Authentication*"
  - Each user and service shares a long-term secret key with the trusted KDC
  - KDC generates a short-term session key (symmetric key) and securely distributes it to the parties that wish to communicate
  - Communicating parties prove to each other that they know the session key

Mediated Authentication

- Goal: two parties eventually share a secret symmetric key ($K_{A\text{-}B}$) for communication
  - Communication partners have keys to contact the KDC ($K_{A\text{-}KDC}$, $K_{B\text{-}KDC}$)
  - Two nonces, “R1” and “R2” are used to proof authenticity



“Alice” has a key $K_{A\text{-}KDC}$ to communicate with KDC

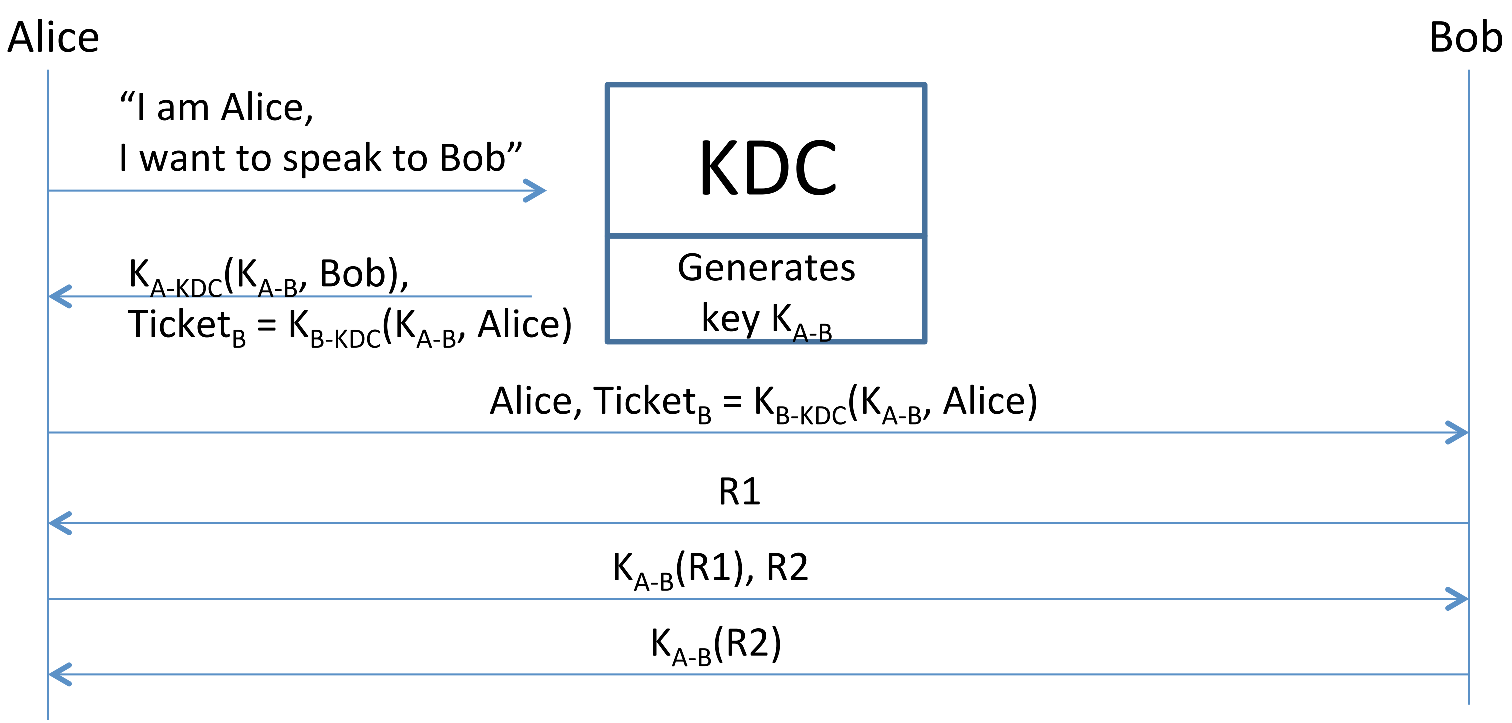
“Bob” has a key $K_{B\text{-}KDC}$ to communicate with KDC

Nonce

- “Nonce” is “Number used once”
- Used in security protocols to
  - Ensure sequence of a set of messages
  - Ensure freshness of a message
- Can be a time stamp, a random number or a counter value
- Should be difficult to guess
- Creators must remember their nonces

Mediated Authentication (2)

- Goal: Avoid KDC having to communicate with Bob
  - Use "Tickets" (security tokens that expire)



Mediated Authentication (2)

- Accessing Services
  - A client application ("Alice", used by an authenticated user) sends a request to the KDC, indicating that it wants to use a particular service ("Bob")
  - The KDC authenticates the client, checks access privileges to service, generates a random symmetric (short-term) session key K_{A-B} for communication between client and server
  - The KDC sends a message back to the client, encoded with the shared key K_{A-KDC} :
    - the value of K_{A-B}, and a **ticket** for accessing the service
  - The client sends the ticket to the service,
  - The service decrypts the ticket, using the secret key K_{B-KDC} , with that it will learn about the session key K_{A-B}
  - Client and service exchange nounces, encrypted with seckert key
    - This is the **mutual authenticator**.

Kerberos – Mediated Authentication

- Widely adopted and implemented in popular operating systems
  - See http://www.kerberos.org
- Kerberos implements mediated authentication with tickets
- Kerberos uses time stamps as "nonces" in the mutual authentication phase of the protocol
- It aims to provide a universal "single sign-on" to services within a network