*Credit to Andrew Donegan & Rory Hughes for sharing the filled-in lecture slides.*

01 Transport Layer

Transport services and protocols provide **logical communication** between processing running different hosts:

1. **Sender side**:  
   Breaks messages into segments and passes them to the **network** layer.
2. **Receiver side**:  
   Reassembles segments into messages and passes them to the **application** layer.

There are multiple transport protocols:

* UDP - DNS / VoIP
* TCP - Email

### Internet Transport Protocols

**TCP** - Reliable, in-order delivery:

* Congestion control.
* Flow control.
* Connection setup.

**UDP** - Unreliable, unordered delivery.

### Multiplexing & Demultiplexing

**Multiplexing at sender**:

Handles data from multiple sockets and adds the transport header.

**Demultiplexing at receiver**:

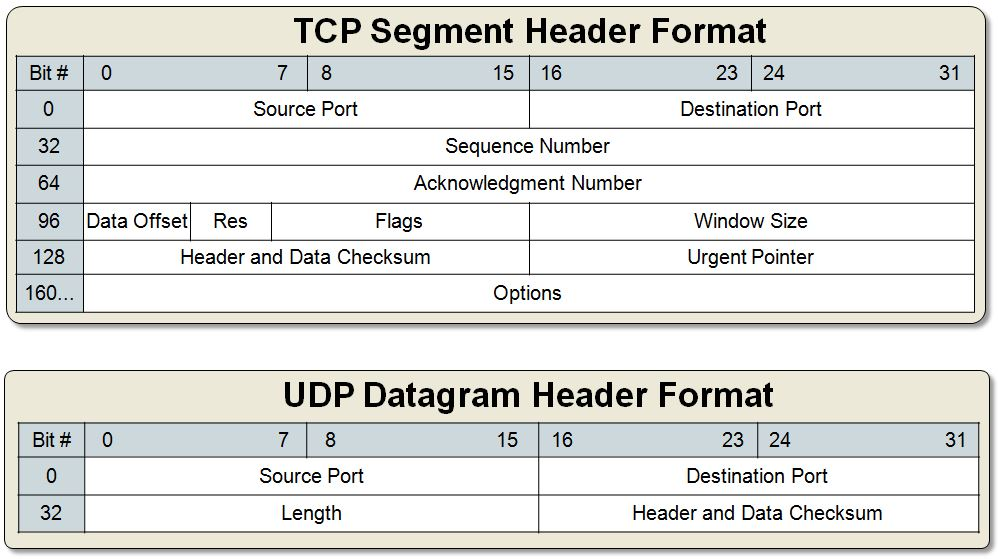
Uses the transport header to deliver received segments to the correct socket.

* Each datagram has:
  + Source IP address
  + Destination IP address
* Each datagram carries one transport-layer segment, which has:
  + Source port
  + Destination port

A typical host has a **MAC**, **IP** address and **port** address.

Non-persistent HTTP will have a different socket for each request.

Why use threads? Because they are lightweight subprocesses.



**TCP**: 20-60B

**UDP**: 8B

# User Datagram Protocol (UDP)

UDP segments may be:

* Lost
* Delivered out-of-order

UDP is **connectionless**, so there is no handshake between the sender and receiver. Each UDP segment is handled **independently** of the others.

UDP is used for:

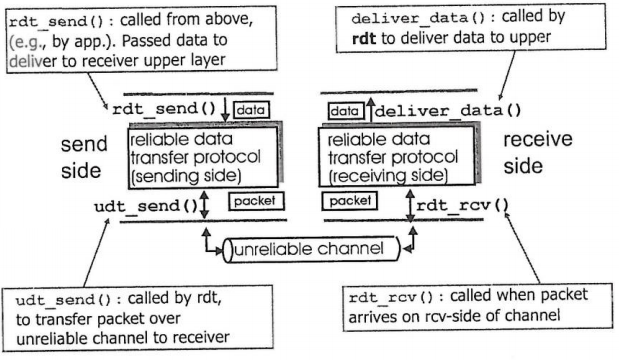
* Streaming multimedia - loss tolerant, rate sensitive
* DNS (Domain Name System)
* SNMP (Simple Network Management Protocol)

UDP has a **small header** size.

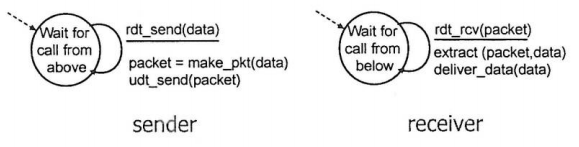
# Reliable Data Transfer (RDT)

Important in the **application**, **transport** and **link** layers.

The characteristics of the unreliable channel will determine the **complexity** of the RDT protocol.



### RDT 1.0

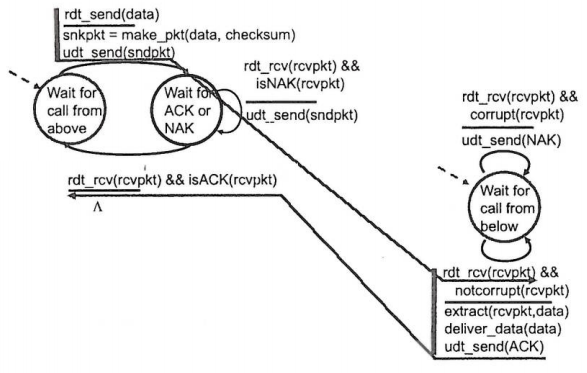


### RDT 2.0

Use a **checksum** to detect bit errors.

Use packet **acknowledgements (ACKs)**.

**Negative acknowledgements (NAKs)** explicitly tell the sender that the packet had errors. The sender resubmits the packet.

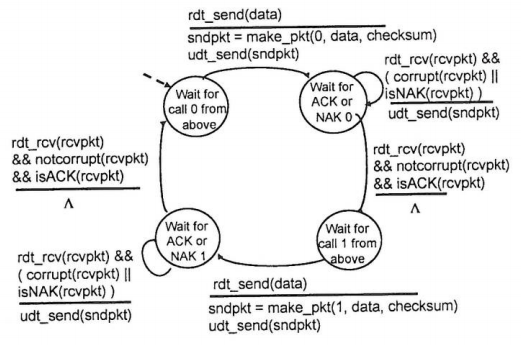


**If ACK / NAK is corrupted, the sender does not know what happened at the receiver**.

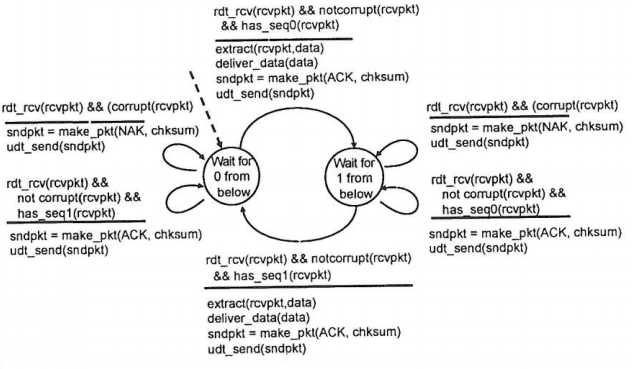
### RDT 2.1

|  |  |
| --- | --- |
| **Sender** | **Receiver** |
| Adds a **sequence number** to the packet (0/1)  Checks if the received ACK / NAK is corrupt | Checks if the packet is a **duplicate** |

**Sender handles garbled ACK / NAKs**:



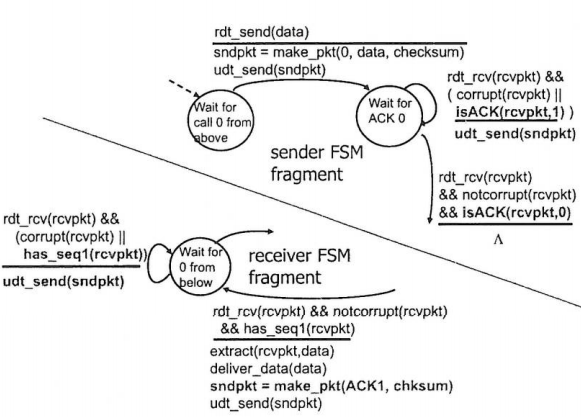
**Receiver handles garbled ACK / NAKs:**



### RDT 2.2

**NAK-free protocol**:

* Use ACKs only.
* Instead of a NAK, receiver sends an ACK for last packet successfully received, along with its **sequence** number.
* A duplicate ACK results in the sender re-sending the packet.

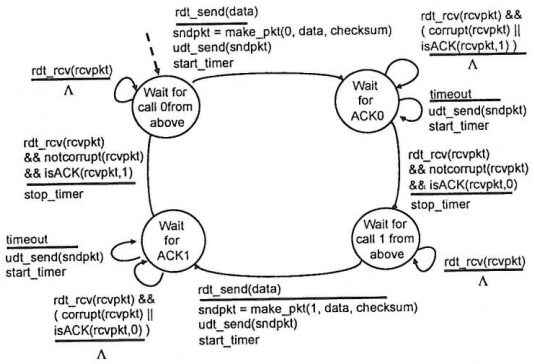


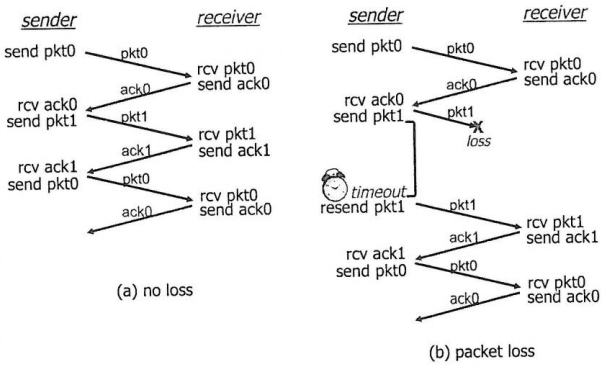
### RDT 3.0

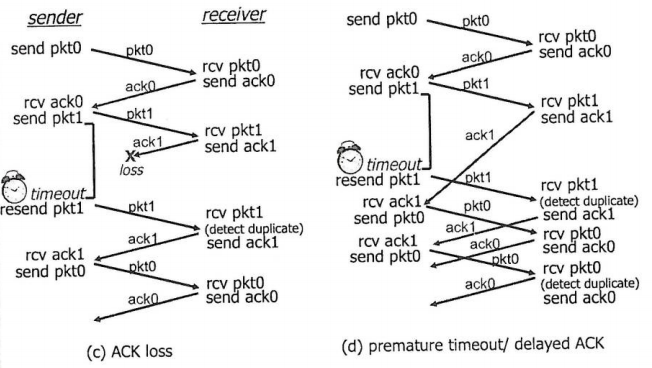
Add a packet **timeout** to account for lost packet.

The packet is **retransmitted** if no ACK is received before the timeout.

**RDT 3.0 Sender**:







### Pipelined Protocols

Pipelining allows the sender to have multiple “in-flight”, yet-to-be-acknowledged packets.

The range of sequence numbers must be increased - buffering at sender and receiver.

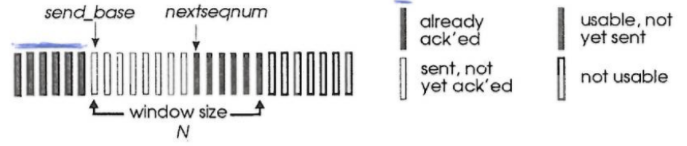
The sender can have up to N unACKed packets in the pipeline.

|  |  |
| --- | --- |
| **Go-Back-N** | **Selective Repeat** |
| Receiver can send cumulative ACKs- Send ACK of the **last received** packet. | Receiver send an individual ACK for **each** packet, even it has already been received. |
| Sender has a timer for the **oldest** unACKed packet - retransmit **all** unACKed packets. | Sender maintains a timer for **each** unacked packet - retransmit only that packet. |

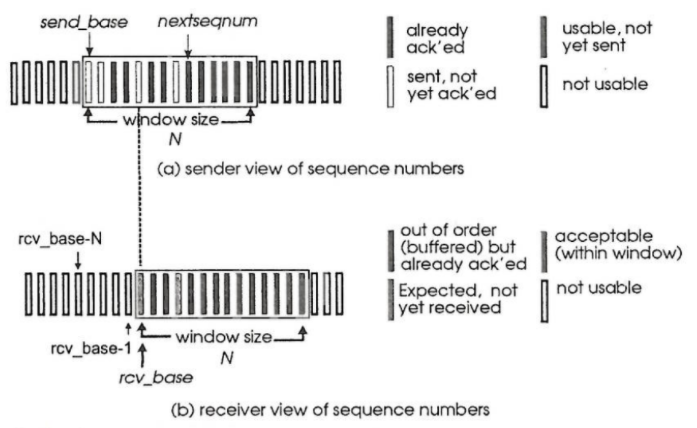
### Go-Back-N

Packet header has a k-bit sequence number.

The window can have up to N consecutive unACKed packets.



### Selective Repeat



# Transmission Control Protocol (TCP)

TCP is:

* **Point to point**: One sender & one receiver.
* **Reliable**: Offers in order byte stream.
* **Pipelined**: TCP congestion and flow control sets the window size.
* **Full-duplex**: Bidirectional data flow in the same connection.
* **Connection-oriented**: Uses handshaking to initialise the sender and receiver state.
* **Flow-controlled**: The sender will not overwhelm the receiver.

Uses sequence numbers and acknowledgements.

The sequence number in TCP refers to the first byte in the segment, and is the current index in the data being sent.

**Piggybacking**: The receiver also echoes back the data received.

### Round-Trip Time (RTT)

The TCP timeout must be greater than the RTT, but RTT varies.

Sample the RTT by measuring time from segment transmission to acknowledgement.

Typically

* If α is too close to zero, sudden changes in the network load will not get reflected in the estimated RTT for a long time.
* If α is too large (close to 1), transient fluctuations in network load will affect the estimated RTT and make it unstable when it should not.

where 4 is the safety margin.

For TCP, the initial value of the timeout interval is **one second**. This value is **doubled** when a timeout occurs.

### Reliable Data Transfer

Retransmission is triggered by:

1. Timeouts (By either lost ACKs or premature timeouts)
2. Duplicate ACKs

TCP is **both** a GBN and SR protocol.

### TCP Fast Retransmit

If sender receives **three ACKs** for the same data, resend the unACKed segment with the **smallest sequence number** - don’t wait for timeout.

### TCP Flow Control

The receiver controls the sender to prevent the sender from overflowing the receiver’s buffer.

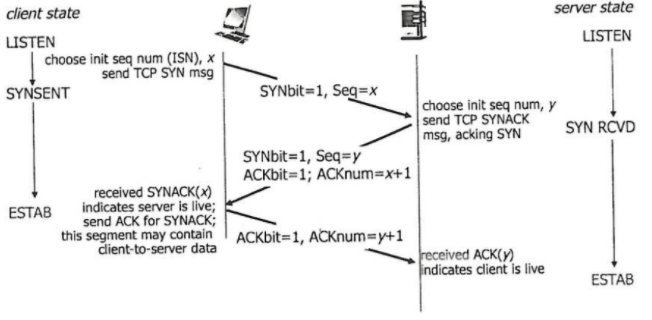
The receiver signals free buffer space by including the (receive window) value in the TCP header of the receiver-to-sender segments. The default is 4096 bytes and can be set via socket options.

The sender limits the amount of in-flight data to the receivers value. This guarantees that the receiver buffer will not overflow.

### Connection Management

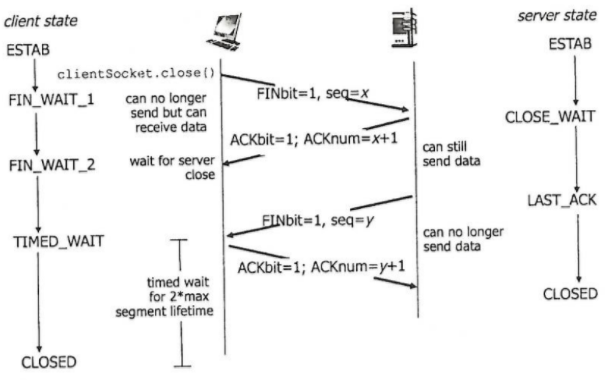
Before exchanging data, the sender and receiver initiate a **3-way handshake** in order to:

* Agree to establish connection.
* Agree on connection parameters.



SYN = Synchronise

### Closing a Connection



### SYN Flood Attack (Denial of Service - DoS)

An attacker sends a large number of TCP SYN segments without completing the third handshake step.

### SYN Cookies

The server does not know if a SYN segment is coming from a legitimate user.

The server creates an **initial sequence number (ISN)** or **cookie** from the **hash** of:

* Source IP address & port number
* Destination IP address & port number
* Timestamp

The server sends a SYNACK but maintains no state information corresponding to the SYN. It then drops the connection.

A legitimate client will return an ACK segment, using the cookie information (ISN+1) in the ACK. The connection is then established.

# Congestion Control

Congestion control prevents too many sources from sending too much data to the server.

This can cause:

* Buffer overflows at routers - Lost packets.
* Queuing in router buffers - Long delays.

Alternate approaches:

1. **End-to-End Congestion Control**:(Used by **TCP**)
   * No explicit feedback from the network.
   * Congestion is inferred from end-system observed loss and delay.
2. **Network-Assisted Congestion Control**:
   * Routers provide feedback to end systems.
   * Single bit indicating congestion (ATM).
   * Explicit rate for sender to send at.

### ATM ABR Congestion Control

**ATM**: Asynchronous Transfer Mode

**ABR**: Available Bit Rate

**ABR** is an elastic service:

* If the path is **underloaded**, the sender should use all **available bandwidth**.
* If **congested**, the sender is throttled to the **minimum guaranteed rate**.

**RM (Resource Management) Cells**:

Sent by the sender, interspersed with data cells.

Bits in the RM cell are set by switches (network assisted):

* **NI bit**: No increase in rate - mild congestion.
* **CI bit**: Congestion indication.

RM cells are **returned to the** **sender** by the receiver with bits intact.

The RM cell contains a two-byte **explicit rate (ER)** field:

* The congested switch may lower the ER value in the cell.
* This is the sender’s send rate (maximum supportable rate on the path).

The **Explicit Feedback Control Indicator (EFCI)** bit in the **data cells** is set to 1 in a congested switch. If the data cell preceding the RM cell has the EFCI bit set, the receiver sets the CI bit in the returned RM cell.

### Additive Increase, Multiplicative Decrease (AIMD)

The sender increases the transmission rate (window size), probing for usable bandwidth until loss occurs.

**Additive Increase**:

Increase the congestion window (cwnd) by one MSS (maximum segment size) every RTT until loss is detected.

**Multiplicative Decrease**:

Cut the cwnd in half after loss occurs.

### TCP Congestion Control

The sender limits transmission:

1. Send cwnd bytes - a function of network congestion i.e. dynamic.
2. Wait one RTT for ACKs.
3. Send more bytes.

### TCP Slow Start

When the connection begins, increase the rate exponentially until the first loss event.

* Initially cwnd = 1 MSS.
* cwnd is incremented for every ACK received i.e. cwnd doubles every RTT.

### Detecting and Reacting to Loss

Loss is indicated by timeout. cwnd is then set to 1 MSS.

The window grows exponentially to the threshold (ssthresh) and then grows linearly.

**TCP Reno**:

Loss is indicated by 3 duplicate ACKs. cwnd is **cut in half** and the window then grows linearly.

**TCP Tahoe**:

cwnd is always **set to one** when a timeout or 3 duplicate ACKs occur.

### Switching from Slow Start to Congestion Avoidance (CA)

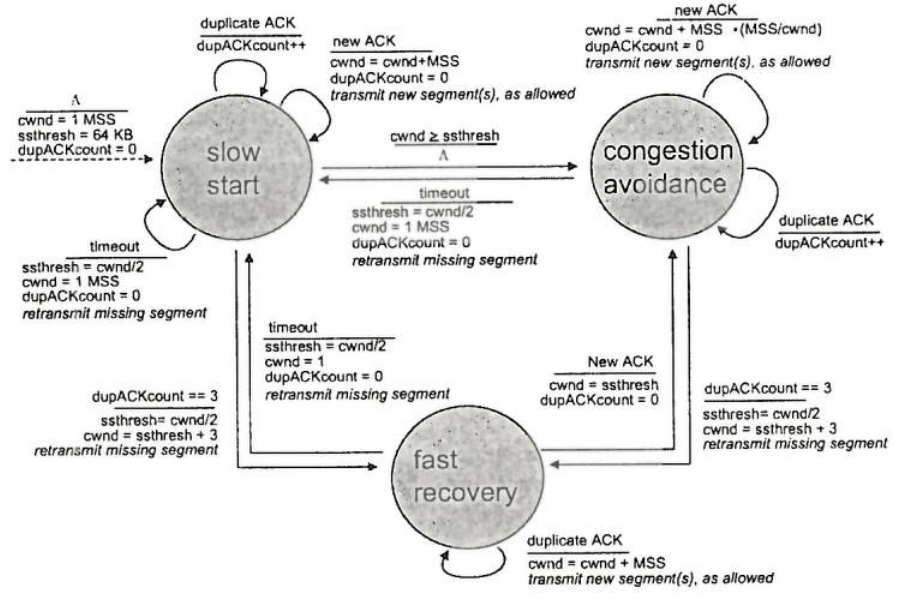
When cwnd gets to half of its value before timeout, exponential increase should switch to linear.

cwnd is increased by 1 MSS every RTT.

Implementation:

* Variable ssthresh.
* On a loss event, ssthresh is set to cwnd/2 just before the loss event.

### TCP Congestion Control



02 Application Layer

### Client-Server Architecture

**Server**:

* Always-on host.
* **Permanent IP** address.
* Data centers for scaling.

**Clients**:

* Communicate with the server.
* May be intermittently connected.
* May have **dynamic IP** addresses.
* Do **not** communicate **directly** with each other.

### Peer-to-Peer Architecture

No always-on server.

**Peers**:

* Request service from other peers.
* Provide service to other peers.
* **Self-scalability**: New peers bring new **service capability** as well as **service demands**.
* Intermittently connected and change IP addresses - **complex management**.

### Processes

A process is a program running within a host.

* Within the **same** host, two processes communicate using **interprocess communication** defined by the OS.
* Processes in **different** hosts communicate by **exchanging messages**.

A **client process** is a process that initiates communication.

A **server process** is a process that waits to be contacted.

Applications with P2P architectures have **client processes** and **server processes**.

### What Transport Services does an App Need?

|  |
| --- |
| **Data Loss**:  Some apps require 100% reliable data transfer (e.g. file transfer).  Other apps can tolerate some loss. |
| **Throughput**:  Some apps (e.g. multimedia) require a minimum amount of throughput to be effective.  Other apps (elastic apps) make use of whatever throughput they can get. |
| **Timing**:  Some apps (e.g. interactive games) require low delay to be effective. |
| **Security**:  Encryption, authentication, data integrity etc. |

### The World Wide Web (WWW)

A web page consists of objects (e.g. HTML file, JPEG). Each object is addressable by a **Uniform Resource Locator (URL)**.

### Hyper Text Transfer Protocol (HTTP)

HTTP uses **TCP**:

1. Client create a TCP connection (socket) to the server on **port 80**.
2. The server accepts the connection from the client.
3. HTTP messages are exchanged between the web browser and the web server.
4. TCP connection is closed.

HTTP is **stateless** - the server maintains no information about past clients’ requests because TCP does it.

### HTTP Connections

|  |
| --- |
| **Non-Persistent HTTP**:  At most **one object** is sent over a TCP connection. The connection is then **closed**.  Downloading multiple objects requires **multiple connections**. |
| **Persistent HTTP**:  **Multiple objects** can be sent over a single TCP connection between the client and server. |

By default, HTTP uses **persistent connections** with pipelining.

### Response Times

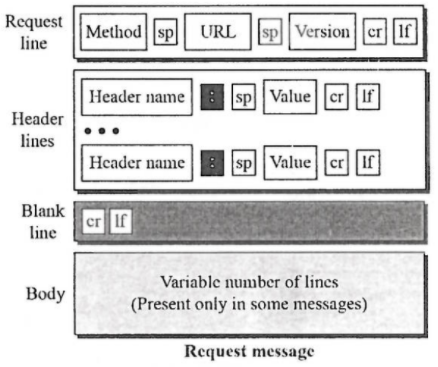
|  |  |
| --- | --- |
| **Non-Persistent HTTP** | **Persistent HTTP** |
| Two RTTs required per object.  OS overhead for each TCP connection.  Browsers often open **parallel** TCP connections to fetch referenced objects. | Server leaves connection open after sending a response.  Subsequent HTTP messages are sent over this connection.  The client sends requests as soon as it encounters a referenced object. |

### HTTP

HTTP messages:

1. Request
2. Response

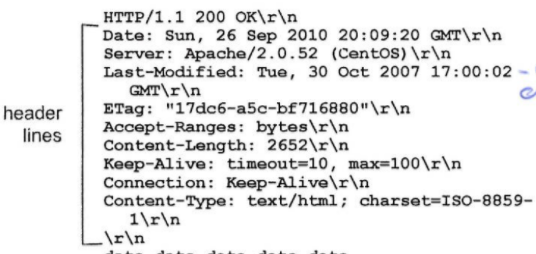
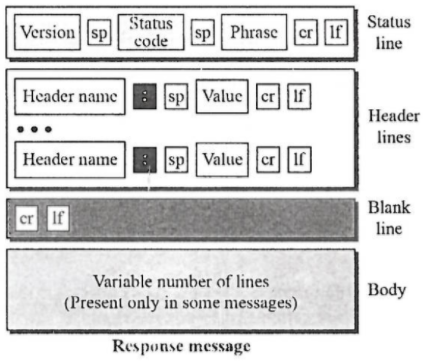
**HTTP request message**:



HTTP method types:

* GET
* POST
* HEAD
* PUT - Uploads file in entity body to path specified in URL field.
* DELETE - Deletes the file specified in the URL field.
* POST - Input is uploaded to the server in entity body.
* URL - Uses GET method. Input is uploaded in URL field of request line.

**HTTP response message**:



### HTTP Response Codes

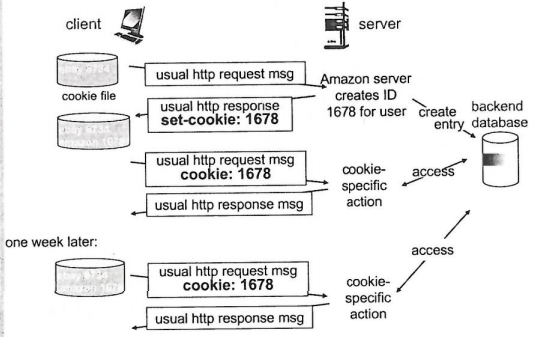
Sample codes:

* **200** OK
* **301** Moved Permanently
* **400** Bad Request
* **404** Not Found
* **505** HTTP Version Not Supported

### Cookies

Cookies can be used for:

* Authorisation
* Shopping carts
* Recommendations
* User session state (webmail)



Cookies can be used to allow HTTP messages to carry **state**.

### Web Caches (Proxy Server)

Used to satisfy client requests without involving the origin server.

The browser sends all HTTP requests to the proxy server:

* If the object is cached - proxy returns the object.
* Otherwise the proxy requests the object from the origin server - proxy returns the object.

The cache acts as both a client and server:

* Server for original requesting client.
* Client to origin server.

Web caching **reduces the response time** for client requests, and **traffic** on access links.

### Conditional GET

The cache specifies the date of the cached copy in a HTTP request.

If the server response contains no object, then the cached copy is up-to-date.

# Electronic Mail

Three major components:

1. User Agents
2. Mail Servers
3. Simple Mail Transfer Protocol (SMTP)

### Mail Servers

A **mailbox** contains **incoming** messages for the user.

A **message queue** contains **outgoing** mail messages.

SMTP protocol is used to send email messages **between mail servers**.

### SMTP

Uses TCP to reliably transfer email messages from the client to the server on **port 25**.

Messages are in **7-bit ASCII** format.

Direct transfer - Sending server to receiving server.

Three phases of transfer:

1. Handshaking (greeting)
2. Transfer of messages
3. Closure

Uses a **command / response interaction** (like HTTP, FTP)

**Commands**: ASCII text

**Response**: Status code and phrase

SMTP process:

1. User agent sends message to their mail server - placed in queue.
2. Client side of SMTP opens TCP connection with recipient’s mail server.
3. SMTP client sends sender’s message over TCP.
4. Recipient’s mail server places the message in the recipient’s mailbox.
5. Recipient invokes their UA to read the message.

### Mail Message Format

**RFC 822** is the standard for text message format:

* Header lines:
  + To:
  + From:
  + Subject:
* Body:
  + ASCII characters.

### SMTP vs. HTTP

|  |  |
| --- | --- |
| **HTTP** | **SMTP** |
| Pull | Push |
| Each object is encapsulated in its own response message. | Multiple objects in a single message. |

Both have **ASCII command / response** interaction and **status codes**.

### Multipurpose Internet Mail Extension (MIME)

MIME is a supplementary protocol that allows **non-ASCII data** to be sent via **SMTP**.

It converts non-ASCII to ASCII and vice versa.

### Mail Access Protocols

|  |
| --- |
| **SMTP**:  Delivery / storage to receiver’s server. |
| **Mail Access Protocol**:  Retrieval from server. |
| **Post Office Protocol (POP)**:  Authorisation, Download. |
| **Internet Mail Access Protocol (IMAP)**:  More features, including manipulation of stored messages on the server. |
| **HTTP**:  Gmail, Hotmail etc. |

### POP3 Protocol

Authorisation phase:

* Client commands:
  1. user: Declare username
  2. pass: Password
* Server responses:
  1. +OK
  2. -ERR

Transaction phase, client:

* list: Lists message numbers
* retr: Retrieve message by number.
* dele: Delete
* quit

|  |  |
| --- | --- |
| **POP3** | **IMAP** |
| Two modes:   1. Download and delete 2. Download and keep   Stateless across sessions. | Keeps all messages in one place on the server.  Allows users to organise messages in folders.  Keeps user state across sessions. |

# Domain Name System (DNS)

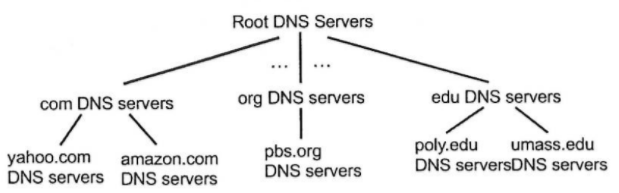
DNS is a distributed database implemented in a hierarchy of many **name servers**.

DNS is an **application-layer** protocol. Hosts and name servers communicate to resolve names.

DNS services:

* Hostname to IP address translation.
* Host aliasing.
* Canonical, alias names.
* Mail server aliasing.
* Load distribution.
* Replicated Web servers.

Many IP addresses correspond to one name.



### Root Name Servers

Root name servers are contacted by a **local name server** that cannot resolve a name.

The root name server returns a list of IP addresses for responsible **TLD servers**.

### Top Level Domain (TLD) & Authoritative Servers

**TLD servers** are responsible for com, org, net, edu, aero, jobs, museums and all top-level country domains.

**Authoritative DNS** servers organise their own DNS servers, providing authoritative hostname to IP mappings for the organisations’ named hosts. They can be maintained by an organisation or service provider.

### Local DNS

Local DNS does not strictly belong in the hierarchy. Each ISP has one. This is also called the **default name server**.

When a host makes a DNS query, the query is sent to its local DNS server, which has a local cache of recent name-to-address translation pairs (may be out of date). It also acts as a proxy and forwards queries to the hierarchy.

### DNS Name Resolution & Caching

An **iterated query** returns the name of the server to contact for name resolution.

A **recursive query** puts the burden of name resolution on the contacted name server.

Once a name server learns of a mapping, it **caches** it:

* Cache entries **time out** after some time - TTL (Time to live).
* TLD servers are typically cached in local name servers.
* Root name servers are not often visited.

Cached entries may be out of date. If the host changes IP address, this may not be known Internet-wide until all TTLs expire.

### DNS Resource Records (RRs)

RR format:



**A**: IPv4 address record.

**AAAA**: IPv6 address record (128-bit).

**NS**: Name Server record.

**CNAME**: Canonical Name (alias) record.

**MX**: Mail Exchange record.

### A & AAAA Records

IPv4 address record:



IPv6 address record:



### Name Server (NS) Records

Stores the hostname of the **authoritative server** for a domain.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **TTL** | **Type** | **Data** |
| ns.example.com | 1800 | A | 192.168.1.2 [Local Domain - never use this] |
| example.com | 1800 | NS | ns.example.com |

### Canonical Name (CNAME) Record

Stores the canonical (real) name associated with the name.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **TTL** | **Type** | **Data** |
| www.example.com | 1800 | A | 192.168.1.2 |
| ftp.example.com | 1800 | CNAME | www.example.com |

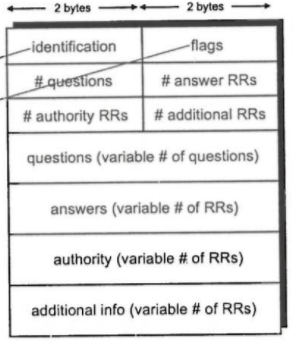
### Mail Exchange (MX) Record

Stores the name of the **mail server** associated with name.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **TTL** | **Type** | **Data** | **MX Level** |
| mail1.example.com | 1800 | A | 192.168.1.2 |  |
| mail2.example.com | 1800 | A | 192.168.1.4 |  |
| example.com | 1800 | MX | mail1.example.com | 10 |
| example.com | 1800 | MX | mail2.example.com | 20 |
| example.com | 1800 | MX | mail100.backupexample.com |  |

### DNS Protocol & Messages

Query and reply messages both have the **same message format**.



Message header:

* **Identification**:
  + 16-bit number for a query.
  + The reply uses the same number.
* **Flags**:
  + Query or reply
  + Recursion desired
  + Recursion available
  + Reply is authoritative
* **Questions**:
  + Name & type fields for a query.
* **Answers**:
  + RRs in response to the query.
* **Authority**:
  + Records for authoritative servers.

### Inserting Records into DNS

1. **Register the new name at a DNS registrar**:  
   Provide names and IP addresses of the **authoritative name server** (primary and secondary).  
   Registrar inserts two RRs into .com TLD server:
   * [website.com, NS, dns1.website.com]
   * [dns1.website.com, A, 212.212.212.1]
2. **Create**:
   * Authoritative server type **A record** for www.website.com.
   * Type **MX record** for website.com

### Attacking DNS

Responses come back in a **single UDP packet** to be quick and to avoid complications.

**DDoS Attacks**:

Bombard the root servers with traffic - Not successful to date, as local DNS servers cache IPs of TLD servers, allowing root server bypass.

**Redirect Attacks**:

*Man-in-the-middle*: Intercept queries.

*DNS poisoning*: Send bogus replies to DNS server which caches them.

# Pure P2P Architecture

Peers are intermittently connected & change IP addresses.

Used for:

* File distribution
* Streaming
* VoIP

### Client-Server vs. P2P

How much time to distribute a file from one server to N peers?

Peer upload / download speed capacity is a limited resource.

### File Distribution Time

|  |  |
| --- | --- |
| **Client-Server** | **Peer-to-Peer** |
| **Server transmission**:  Must sequentially send file copies:   * Time to send one copy = * Time to send copies =   **Client**:  Each client must download the file copy:   * Min client download rate = * Min client download time =   = File size  = File upload speed  Time to distribute to clients:  increases linearly in . | **Server transmission**:  Must upload at least one copy:   * Time to send one copy =   **Client**:  Each client must download the file copy:   * Min download time =   **Clients**:  As aggregate must download bits:   * Max upload rate =   Time to distribute to clients:  increases linearly in , but so does as each peer brings server capacity. |

### BitTorrent

File is divided into **256KB chunks**. Peers in the torrent send / receive these chunks.

**Torrent**: A group of peers exchanging chunks of a file.

**Tracker**: A server that **tracks the peers** participating in torrent.

A peer joining a torrent:

* Has no chunks, but will accumulate them over time from other peers.
* Register with the **tracker** to get a list of peers and connects to a **subset of peers** called **neighbours**.

While downloading, a peer uploads chunks to other peers.

Peers may change peers with whom it exchanges chunks.

**Churn**: Peers may come and go.

Once a peer has the entire file, it may leave or remain in the torrent.

**Requesting Chunks**:

Alice obtains a list of peers from the tracker and begins exchanging file chunks.

At any given time, different peers have different subsets of file chunks.

* Periodically, Alice asks each peer for a **list of chunks** that they have.
* Alice requests missing chunks from peers, **rarest first**.

**Sending Chunks**:

Alice sends chunks to the **four peers** currently sending her chunks at the highest rate. These are re-evaluated every **10 seconds**. Other peers are choked by Alice.

Every 30 seconds, randomly select (**optimistically unchoke**) another peer and start sending chunks. The newly chosen peer may join the top four.

# Web Applications

**Rich Internet Applications (RIAs)** are web applications that approximate the look, feel and usability of desktop applications.

Key attributes:

1. Performance
2. Rich GUI

### Asynchronous Javascript and XML (AJAX)

RIA performance comes from AJAX, which uses client-side scripting to make web applications more responsive.

While a **synchronous** request is being processed on the server, the user cannot interact with the client web browser.

**Asynchronous web communication** allows the user to interact with the page while data loads. The communication pattern is made possible by AJAX.

Typical AJAX request:

1. The user clicks, invoking an event handler.
2. The handler’s code creates an XMLHttpRequest (XHR) object.
3. The XHR object requests a page from the server.
4. The server retrieves the appropriate data and sends it back.
5. The XHR object fires an event when the data arrives.
6. The callback event handler processes the data and displays it.

The XHR object:

* Resides on the client.
* Is a layer between the client and server that manages asynchronous requests in AJAX applications.

For security purposes, the XHR object does **not** allow a web application to request resources from domains **other** than the one that served the application - **Same Origin Policy (SOP)**.

With AJAX, the server has no method of pushing messages to the client.

### WebSockets

The WebSocket specification defines an API establishing a **two-way socket connection** between a web browser and server. Both parties can send data at **any time**.

The client establishes a WebSocket connection through the WebSocket handshake:

* The client sends a regular **HTTP request** to the server.
* The **upgrade header** is included in the request, which informs the server that the client wishes to establish a WebSocket connection.

WebSocket messages have a **two-byte framing overhead**. This frame-based system helps to reduce the amount of non-payload data that is transferred and latency.

03 Network Security

Network security involves:

1. **Confidentiality**:  
   Only the sender and the intended receiver should be able to read a message’s contents.
   * The sender encrypts the message.
   * The receives decrypts the message.
2. **Authentication**:  
   The sender and receiver want to confirm the identity of each other.
3. **Message Integrity**:  
   The sender and receiver want to ensure that the message was not altered, in transit or after, without detection.

Types of attacks:

1. **Passive Attack**:  
   Eavesdrop or intercept messages.
2. **Active Attack**:  
   Actively insert messages into the connection.
3. **Impersonation**:  
   Fake the source address in a packet (or any other field).
4. **Hijacking**:  
   Take over the ongoing connection by removing sender or receiver, inserting themselves in place.

# Cryptography

The original data to be transferred is called **plaintext (P)** or cleartext.

The encrypted version is called **ciphertext (C)**.

An **encryption function** operates on P to produce C.

In the reverse process, a **decryption function** D operates on C to produce P.

The following identity must hold true for the cryptosystem to function correctly:

### Cryptographic Keys

All modern encryption algorithms use a **key (K)**.

The key can take on a range of possible values - the **key space**.

The encryption and decryption functions now become:

and

### Caesar Cipher

The Caesar Cipher is a **monoalphabetic substitution**.

Attacks:

1. Commonly occurring characters.
2. Commonly occurring bigrams (e.g. th, he, in, er).
3. Domain-specific buzz words (e.g. system, login, password).

### The Vigenѐre Cipher

Some protection from the previous attacks can be gained using a **polyalphabetic cipher**.

1. Create a 2D matrix of 26 different characters (26\*26).
2. Pick a key (e.g. “AFGHANISTANBANANASTAN”).
3. Use row A to encrypt the first letter, row F the second etc.

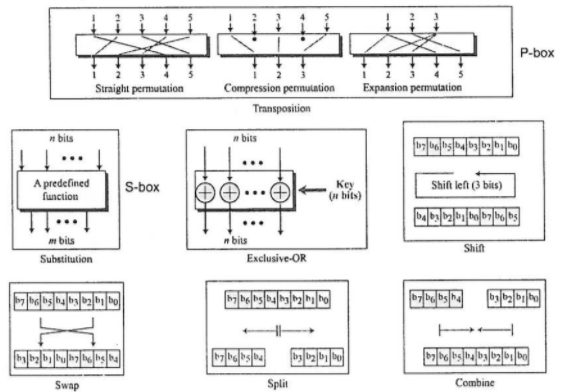
### Transposition Ciphers

Transposition ciphers **reorder** the symbols, but do not disguise them.



The plaintext is written horizontally in rows, and the ciphertext is read out in columns, starting with the column whose key is the lowest.

### Components of a Modern Block Cipher



# Symmetric-Key Encryption

The sender and receiver both use the same:

* Secret key
* Cryptosystem

The sender uses the key to encrypt the message.

The receiver uses the same key to decrypt the message.

Examples:

* DES
* IDEA
* AES

### Key Management

The main problem is getting the sender and receiver to agree on a secret key, without anyone else finding out.

### Data Encryption Standard (DES)

The algorithm is easily implemented in hardware. Software implementations are also widely available.

DES is a **block cipher** that operates on a **single chunk** of data (8 bytes) at a time. It produces an **8 byte** output.

The key length is **56 bits**, often expressed as an 8-character string, with the extra bits used as a **parity check**.

The algorithm has 19 distinct stages:

* The first stage reorders the bits of the 64-bit input block by applying a fixed permutation (P-box).
* The last stage is the exact inverse of this permutation.
* The penultimate stage exchanges the leftmost 32 bits with the rightmost.
* The remaining 16 stages are called **rounds**.
  + They are functionally identical, but their input is a quantity computed from the key and round number.
  + The round key is 48 bits long.

### Cracking DES

56 bits is a short key, so we can brute force - try every key.

* 256 encryptions to try all keys.
* Special chips can crack 4M keys/s.
* A $1M DES cracking machine could break it in a few hours.

*Improvement*: Use Triple DES or 3DES.

* Makes use of two keys (112 bits) or three keys (168 bits).
* Uses EDE or DED mode.

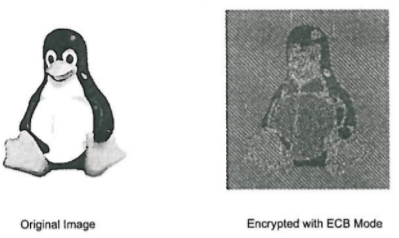
### Modes of Operation for Block Ciphers

DES uses Electronic Code Book (ECB) mode.

* Each 64-bit block is encoded independently of all other blocks. Block synchronisation between the Tx and Rev is not necessary.
* Bit errors from noisy transmission only affect the corresponding blocks.
* Block ciphers operating in ECB can be parallelised - high speed implementations.

### ECB Problems

ECB mode encrypts in a highly deterministic manner. Identical plaintext blocks result in identical ciphertext blocks, as long as the same key is used.



### ECB Substitution Attacks

Each block is encrypted independently of all other blocks, so they can be reordered maliciously.

### Cipher Block Chaining Mode (CBC)

In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This makes each ciphertext block dependent on all previous blocks.

To make each message unique, an initialisation vector (IV) must be used in the first block.



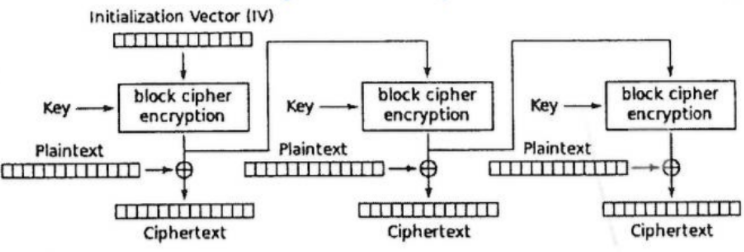
If we encrypt the same plaintext with different IVs, two resulting ciphertext sequences will look completely unrelated to each other.

The IV does **not** need to be secret. It is only used once (a nonce).

### Output Feedback Mode (OFB)

In OFB, a block cipher is used to build a **stream cipher** encryption scheme. The key stream is generated in a **blockwise** fashion instead of bitwise.

The cipher outputs key stream bits, where is the width of the block cipher used. We can then encrypt plaintext bits using the XOR operation.

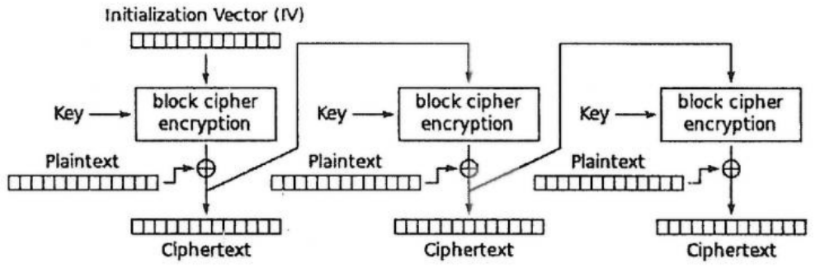


The OFB mode forms a **synchronous stream cipher**, as the key stream does not depend on the plain or ciphertext.

Encryption and decryption are the same operation.  
  
In OFB mode, block cipher computations are independent of the plaintext, so we can pre-compute several blocks of key stream material.

### Cipher Feedback Mode (CFB)

CFB is similar to OFB, except the ciphertext is fed back in as opposed to the output of the block cipher.



CFB is an **asynchronous stream cipher** since the stream cipher output is also a function of the ciphertext.

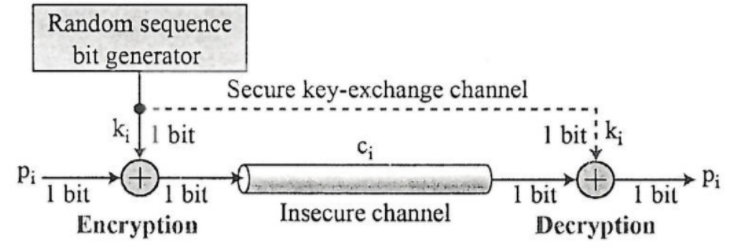
A variant of CFB can be used in situations where short plaintext blocks are to be encrypted.

Plaintext generated by the keyboard is typically only one byte long. In this case, only 8 bits of the key stream are used for encryption, and the ciphertext is a single byte.

### The Vernam Cipher

The simplest and most secure **stream cipher** is the **one-time pad**.

It chooses a key stream (k) that is randomly chosen for each encipherment. It makes use of the XOR operator.

.

k ≥ p

### Advanced Encryption Standard (AES)

AES is a symmetric cipher with variable key and block sizes of 128 (most common), 192 and 256 bits.

It supports fast encryption and decryption in software and can be implemented efficiently in smartcards.

The cipher consists of between 10 and 14 rounds () depending on the key length () and block length ().

A plaintext block undergoes rounds of operations to produce an output block . Each operation is based on the value of the th round key.

Round keys are derived from the cipher key by first **expanding** the key, then selecting parts of the expanded key each round.

# Asymmetric-Key Encryption

In **public key cryptography**, each user generates a pair of keys:

1. **Public key (K+)**: Published and widely distributed.
2. **Private key (K-)**: Kept secret.

This solves the management problem associated with symmetric key cryptosystems.

Examples:

* RSA
* Diffie-Hellman
* ElGamal
* ECC

### Properties

It must be computationally **easy** to **encipher or decipher** a message given the appropriate key.

It must be computationally **infeasible** to **derive the private key** from the public key.

Messages are encrypted with a public key, and decrypted with the corresponding private key.

### Rivest-Shamir-Adleman Encryption (RSA)

RSA’s security is based on the difficulty of **factorising very large numbers**.

**Easy**: Calculate product of two prime numbers.

**Hard**: Calculate prime factors from the product.

### Modular Arithmetic

Most cryptographic algorithms are based on arithmetic with a finite set of numbers.

is congruent to if divides

where:

* = modulus
* = remainder

### Multiplicative Inverse

The integers modulo is denoted .

The **multiplicative inverse** of a modulo is an integer such that:

is the multiplicative inverse of where .

The multiplicative inverse **only exists** for an element iff:

### The Extended Euclidean Algorithm (EEA)

The EEA is a method for computing the greatest common divisor of two integers, and .

It computes integers and such that .

1) Repeatedly divide the divisor by the remainder until the remainder is zero.

**Example**:

2) Reverse the steps of the EEC to find these integers and . Start with the GCD.

The modular multiplicative inverse of a modulo can be found with the EEA.

Back substitution:

### Euler’s Totient Function

is the number of positive integers that are less than , and **relatively prime** to .

and are relatively prime if

1, 3, 7, 9 are relatively prime to 10.

1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20 are relatively prime to 21.

when is **prime**.

### Euler’s Phi Function

Let have the following canonical factorisation:

where:

* are distinct primes.
* are positive integers.

*Hint*:

### Fermat’s Little Theorem

Let be an integer and a prime, then:

We now have a way for inverting an integer modulo a prime .

Compute :

via

# RSA Algorithm

1. Choose two large distinct primes and .
2. Compute the product .
3. Randomly choose an encryption key , less than that has no common factors with .
   * and are relatively prime.
   * is invertible iff .
4. Compute the decryption key such that:

### RSA Usage

The numbers and are the **public key**.

The number is the **private key**.

Break the plaintext message into a number of blocks and represent each block as an integer.

**Encryption**:

**Decryption**:

Key sizes can be 1024, 2048, 3072, 7680 bits. 3072 is considered secure in 2015.

### RSA Example

1. Let and .
2. Using :
3. Choose as and have no common factors.
4. Solving and :  
    Using EEC

Since , the plaintext can be at most five bits long . Each block can only contain a single character.

### Fast Exponentiation

A straightforward way of exponentiation is like this:

Alternatively we can compute:

What about ?

### Square and Multiply Algorithm

Based on scanning the bits of the exponent from the most significant to least significant (L→R).

In every iteration (every exponenent bit):

1. The current result is squared.
2. If (current bit == 1)  
   A multiplication of the current result by is executed following the squaring.

**Example**:

Initial setting - bit processed:

SQ - bit processed:

MUL since

SQ - bit processed:

No MUL since

SQ - bit processed:

MUL since

SQ - bit processed:

No MUL since

# Finding Large Primes

The general approach is to generate integers at random which are then checked for primality.

Chance that a randomly picked integer is prime is .

In practice, we only test odd numbers, so the probability is double.

For RSA with 1024-bit modulus , the primes and should have length 512 bits.

### Primality Tests

A simple primality test can be based on Fermat’s Little Theorem.

However, there are certain composite numbers which may fulfill the above condition. In order to detect them, the algorithm runs times with different values of .

*Is 221 a prime number*?

for test.

should be 0 if prime.

Not prime.

### Miller-Rabin Primality Test

Given the decomposition of an odd prime candidate :

where is odd.

If we can find an integer such that:

and

for all , then is probably a prime.

**Example**:

If then , so .

or or for each from 1 to 12.

*Is 561 prime*?

If then , so .

or or or

### RSA in Practice

RSA encryption is deterministic.

In practice, RSA has to be used with a padding scheme.

**Optimal Asymmetric Encryption Padding (OAEP)** for padding RSA messages are specified. This is standardised in the Public Key Cryptography Standard #1 (PKCS #1).

### RSA Malleability

A crypto scheme is said to be **malleable** if an attacker is capable of transforming the ciphertext into another ciphertext, which leads to a known transformation of the plaintext.

This is easily achievable in the case of RSA if the attacker replaces the ciphertext by , where is some integer:

n

The receiver computes

If were an amount of money to be transferred, choosing could double the amount of money in a way that goes undetectable by the receiver.

### RSA Side Channel Attacks

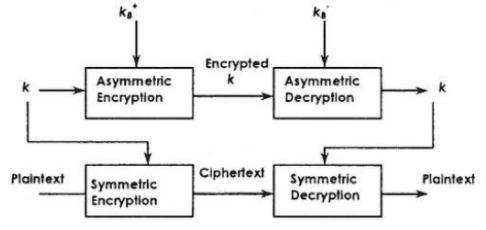
Exploit information about the private key which is leaked through physical channels such as power consumption.

Short and long high-activity intervals are explained by the square and multiply algorithm:

* If an exponent bit has the value 0, just a square is performed.
* If an exponent bit has the value 1, a square and multiply is performed.

### Hybrid Schemes

Asymmetric-key algorithms are **not** a replacement for symmetric-key algorithms such as DES or AES. Rather, they supplement AES or any other fast bulk encryption cipher.



We can use a public key algorithm to securely transfer a session key . This session key can be used for bulk encryption & decryption.

### An Important Property of RSA

The following commutative propery will be useful later:

# Message Authentication & Integrity

**Authentication** validates that a message came from the **original source**.

**Integrity** verifies that the message has **not changed**.

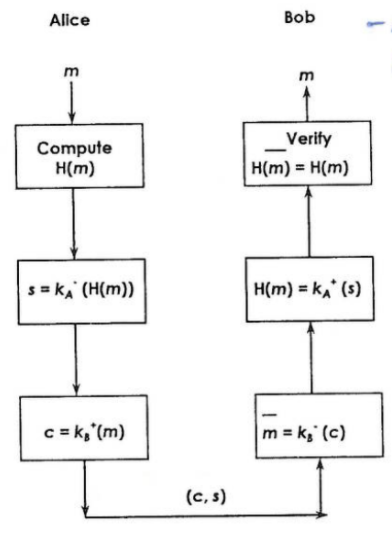
A **message digest / cryptographic hash** is a strong digital fingerprint of a message:

* Takes an input and produces a **fixed-length** value - 128 / 160 / 256 bits.
* Computationally infeasible to find two different messages and such that:
* Examples:
  + MD2, MD4, MD5
  + SHA-1, SHA-2

### Digital Signatures

The sender digitally signs the document establishing that they are the document owner / creator.

The recipient can prove to someone that the sender, and no one else must have signed the document. It is verifiable and non-forgeable (non-repudiation).



### Key Management

When Alice obtains Bob’s public key, how does she know that it’s really his key?

One way is to enlist the services of a **trusted third party (TTP)**.

### X.509 Certificates

The TTP constructs a message referred to as a **certificate**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Subject  (Identity of User) | Public Key | Validity Period | Issuer  (Identity of TPP) | Other Fields | Signature of TTP |

Contains a digital signature on the other fields with the private key of the TTP.

This system assumes that every user is equipped with the public key of the TTP. This allows everyone to verify the digital signature on the certificate, guaranteeing that the public key is associated with the named user.

### Certification Hierarchy

TPPs that issue certificates are referred to as **certification authorities (CAs)**.

The **root CA** issues certificates only to **other** CAs.

Each user of the system needs only the public key of the root CA.

# Diffie-Hellman Key Exchange (DHKE)

DHKE allows strangers to establish a shared symmetric key without having to meet, and without the need for a cryptosystem to be in place.

The idea behind DHKE is:

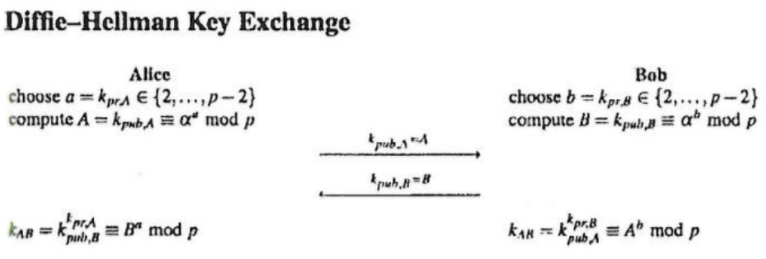
* Exponentiation in the multiplicative group ( is a prime) is a one-way function.
* Exponentiation is commutative:

The value is a joint secret. It can be used as a **session key** between the two parties.

### DHKE Usage

Securely choose the domain parameters:

1. Choose a large prime .
2. Choose an integer .
3. Publish and .



**Exercise**:

Generate a DH key using the domain parameters and .

Assume that and .

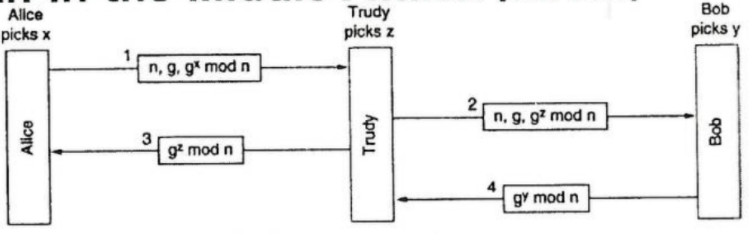
### Security of DHKE

The domain parameter should have a length of 1024 bits or longer.

should be a primitive element or generator of the group , whose powers modulo generate all integers from to . Every element of can be written as:

Is 3 a primitive element modulo ?

### Man-in-the-Middle Attack



Alice computes the key , and so does Trudy for messages to Alice.

Bob computes the key , and so does Trudy for messages to Bob.

Alice thinks she is talking with Bob and establishes a session key with Trudy, as does Bob. This is known as the **bucket brigade attack**.

The **Diffie-Hellman Station-to-Station (STS) protocol** is used to stop MITM attacks.

### Finite Fields

A finite field, also known as a **Galois field**, is a set with a finite number of elements in which we can **add, subtract, multiply and invert**.

A **group** is a **set** with **one operation** and the corresponding **inverse** operation.

* If the operation is addition, the inverse is subtraction.
* If the operation is multiplication, the inverse is division (or multiplication with the inverse element).

### Groups

A group is a set of elements with an operation which combines two elements of and has the following properties:

1. The group operation is **closed**.
2. The group operation is **associative**.
3. There is an element called the **identity element**.
4. For each there exists an element called the **inverse** of .
5. The group is **commutative**.

**Example**:

The set of integers and the operation addition modulo form a group with the neutral element 0. Every element has an inverse such that

This set does **not** form a group with the multiplication operation because most elements do not have an inverse such that:

In order to have all four basic arithmetic operations in one structure, we need a set called a **field**.

### Fields

A field is a set of elements with the following properties:

* All elements of form an additive group with the group operation and the neutral element .
* All elements of except form a multiplicative group with the group operation and the neutral element .
* When the two group operations are mixed, the distributivity law holds:

### Prime Fields

The set (where is a prime) is denoted as and is referred to as a **prime field** a.k.a. a Galois field with a **prime number of elements**.

Elements of the field can be represented by integers :

* All non-zero elements of have an **inverse**.
* Arithmetic in is done **modulo** .  
  e.g.

### Finite Groups

A group is finite if it has a finite number of elements e.g. , .

We denote the cardinality / order of the group by .

2. consists of integers for which .  
     
    i.e. elements

The order of an element of a group is the smallest positive integer such that

**Example**:

We try to determine the order of in . For this, we keep computing powers of until we obtain the identity element .

The pattern repeats after the identity element is found:

### Cyclic Groups

A group which contains an element with maximum order is said to be cyclic.

Elements with maximum order are called primitive elements or generators.

**Exercise**: Check if is a primitive element of .

implies that is a primitive element and that is cyclic.

For every prime , is a commutative finite cycle group i.e. the multiplicative group of every prime field is cyclic.

Let be a finite cycle group:

* The number of primitive elements of is .
* If is prime, then all elements are primitive.

**Example**: Find the number of primitive elements in .

The primitive elements of are .

### Subgroups

Let be a cyclic subgroup. Every element with is the primitive element of a cyclic group with elements

i.e. Any element of a cyclic group is the generator of a subgroup, which in turn is cyclic.

**Example**:

Consider a subgroup of .

which generates the subgroup

More precisely, it is the subgroup of prime order .

is not the only generator of , also .

### The Discrete Logarithm Problem (DLP) in Z\*p

Given a finite cycle group of order and a primitive element and another element , the DLP is determining an integer such that .

is called the discrete logarithm of to the base .

**Exercise**:

Find given the cyclic group in which is a primitive element and .

# The ElGamal Encryption Scheme

The ElGamal encryption scheme can be viewed as an **extension of the DHKE** protocol.

Its security is based on the intractability of the discrete logarithm problem.

We consider the ElGamal encryption scheme over the group , where is a prime.

If Alice wants to send an encrypted message to Bob, both parties first perform a DHKE to derive a shared key , assuming that and have been generated.

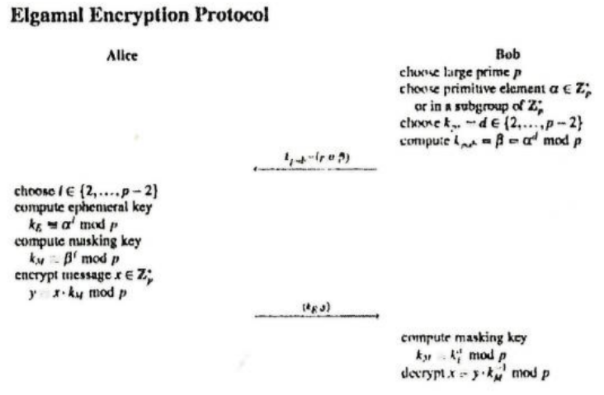
The new idea is that Alice uses as a **multiplicative mask** to encrypt .



Bob computes his private key and public key . This key pair does not change and can be used for encrypting many messages.

Alice however, has to generate a new public-private key pair for the encryption of **every** message. Her private key is and her public key is , the latter of which is an ephemeral (temporary) key.

The joint key is denoted by and is used for masking the plaintext.



The setup phase is executed once by the party who issues the public key and who will receive the message.

The encryption and decryption phases are executed every time a message is sent.

In contrast to DHKE, no TTP is needed to choose a prime and primitive element.

### ElGamal Issues

ElGamal is a **probabilistic** encryption scheme. Encrypting the same message with the same public key will result in two different ciphertexts. The session key used for encryption is chosen at random for each encryption.

The ciphertext consists of two parts:

1. The ephemeral key
2. The masked plaintext

Since all parameters have a bit length of , the ciphertext is twice as long as the message. The **message expansion factor** of ElGamal is two.

### Attacks Against the Discrete Logarithm Problem

The security of many asymmetric primitives is based on the difficulty of computing the DLP in cyclic groups.

To compute for a given and in such that

**Brute-Force Search**:

A brute-force search is the most naive and computationally costly way for computing the discrete logarithm . We simply compute powers of the generator successively until the result equals .

For a random logarithm , we do expect to find the correct solution after checking **half** of all possible . This gives us a complexity of steps, where is the cardinality of the group.

To avoid brute-force attacks on DL-based cryptosystems, the cardinality of the underlying group must be sufficiently large.

In the case of the group where is prime, tests are required on average to compute a discrete logarithm. should be at least 280 bits.

# Elliptic Curve Cryptography (ECC)

ECC is based on the generalised discrete logarithm problem of finding the integer such that:

ECC is more efficient. It requires fewer bits for the same security ⇒ fewer computations.

An elliptic curve over is the set of all pairs which fulfill:

Together with an imaginary point and the condition , the curve is non-singular i.e. it has no self-intersections or vertices.

An elliptic curve has **horizontal symmetry**, i.e. any point on the curve can be reflected over the x-axis and remain in the same curve.

Two points on an elliptic curve can be **added** to produce a **third point**.

### Group Operations on Elliptic Curves

Given two points and and we compute the coordinates of the third point R such that:

**Point Addition**:

Draw a line through and and obtain a third point of intersection between the elliptic curve and the line. Mirror this point along the x-axis to obtain .



### Computations on Elliptic Curves

**Point Addition & Point Doubling Formulae**:

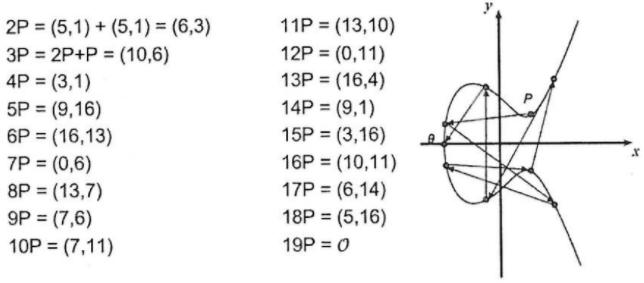
where:

**Example**:

Given and a point , compute .

()

The points on an elliptic curve and the point at infinity form cyclic subgroups.



This elliptic curve has .

### Number of Points on an Elliptic Curve

Determining the point count on elliptic curves is hard.

**Hasse’s Theorem**:

Given an elliptic curve , the number of points on the curve is denoted by and is bounded by:

The number of points is close to the prime . To generate a curve with about 2160 points, a prime with length of about 160 bits is required.

### Elliptic Curve Logarithm Problem

ECC cryptosystems are based on the fact that is large, kept secret, and attackers cannot compute it easily.

Given an elliptic curve . We consider a primitive element and another element .

The DL problem is finding the integer where such that:

If is known, an efficient method to compute the point multiplication is required to create a reasonable cryptosystem.

### Double-and-Add Algorithm for Point Multiplication

Point multiplication is analog to exponentiation in multiplicative groups. In order to do so efficiently, we can directly adopt the square-and-multiply algorithm from RSA. The only difference is that squaring becomes doubling, and multiplication becomes addition.

**Algorithm**:

Input:

* An elliptic curve
* An elliptic curve point
* A scalar with bits

Output:

Initialisation:

|  |
| --- |
| for (i = t-1 downto 0) {  T = T + T mod n  if (di == 1) {  T = T + P mod n  }  return T } |

**Example**:

Consider the scalar multiplication , which has the following binary representation:

Initial setting - bit processed:

DBL - bit processed:

ADD since

DBL - bit processed:

No ADD since

DBL - bit processed:

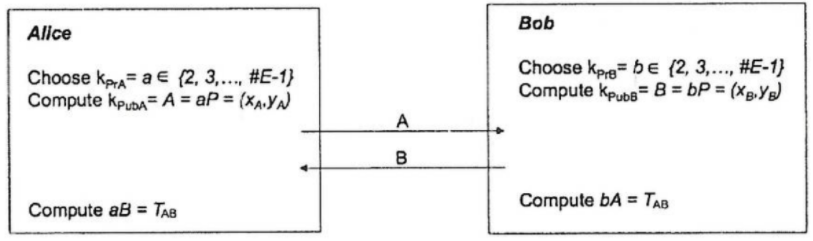
ADD since

DBL - bit processed:

No ADD since

### Elliptic Curve Diffie-Hellman Key Exchange (ECDH)

Given a prime , a suitable elliptic curve and a point , the ECDH key exchange is defined by the following protocol:

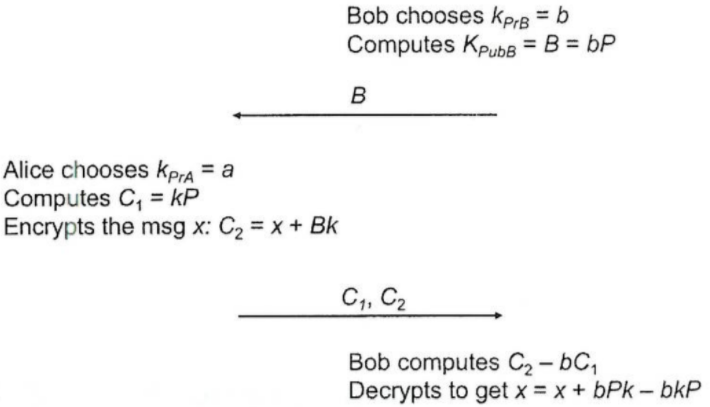


The joint secret between Alice and Bob .

One of the coordinates of the point can be used as a **session key**, often after applying a hash function.

### ElGamal with Elliptic Curves

As before, choose a suitable elliptic curve and a point .



### Key Lengths & Efficiency

256-bit ECC provides the same security as a 3072-bit RSA key.

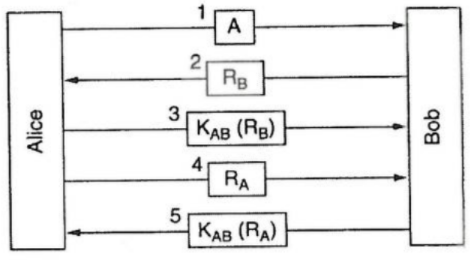
It can be up to **10 times faster**, depending on the implementation.

# Authentication Protocols

Authentication is the technique by which a process verifies that a communicating partner is who it is supposed to be, and not an imposter.

Authentication is based on a **shared secret key** .

The following is an example of a challenge-response protocol, where is a random number used once (a **nonce**):



### The Reflection Attack

|  |  |
| --- | --- |
|  |  |

Trudy can break this protocol if it is possible to open **multiple sessions** with Bob.

### Message Authentication Code from Hash Functions (HMAC)

A HMAC is calculated using a cryptographic **hash function** in combination with a **shared secret key**.



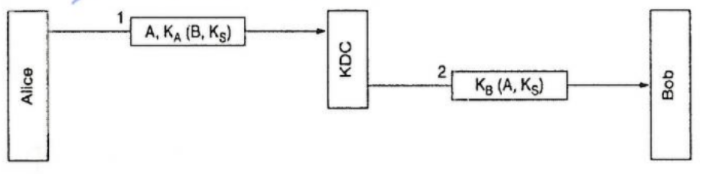
Like digital signatures, they can be used to quickly verify **data integrity** and the **authenticity** of a message. They **do not** provide non-repudiation.

### Authentication Using a KDC

How many keys are required to talk to people using a shared secret?

The alternative is to introduce a **key distribution center (KDC)**.

Each user has a **single shared key** with the KDC. Authentication and session key management happens through the KDC.

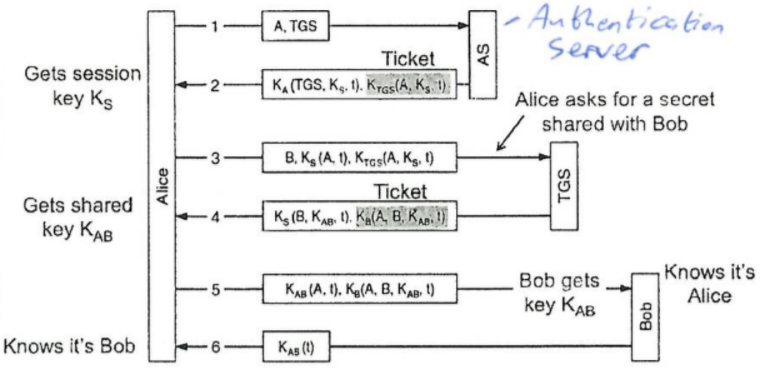


### Needham-Schroder Authentication Protocol

1. Alice tells the KDC that she wants to talk to Bob and sends a message containing a random number as a nonce.
2. The KDC sends back a message containing , a **session key** and a **ticket** that she can send to Bob.
3. Alice sends the **ticket** to Bob along with a **new random number**  encrypted with the session key .
4. Bob sends back to prove to Alice that she is talking to the real Bob.

### Kerberos

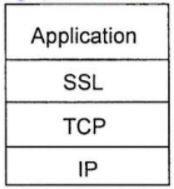
Kerberos V5 is a widely used protocol. Authentication includes a **ticket granting server (TGS)**.



# Secure Sockets Layer (SSL)

SSL provides **transport-layer** security to any **TCP-based application** using SSL services.

In practice, **Transport Layer Security (TLS)** v1.2 should be used.

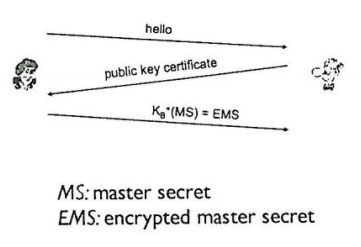


Security services:

* Server authentication
* Data encryption
* Client authentication (optional)

## Toy SSL - A Simple Secure Channel

**Handshake**:

Alice and Bob use their certificates and private keys to authenticate each other and to exchange the shared secret.

**Key Derivation**:

Alice and Bob use their shared secret to derive a set of keys.

**Data Transfer**:

Data to be transferred is broken up into a series of **records**.

**Connection Closure**:

Sends special messages to **securely close** the connection.

### Key Derivation

It is considered bad to use the **same key** for more than one cryptographic operation. Use a **different key** for the Message Authentication Code (MAC) and for encryption.

Four keys are required:

1. The encryption key for data sent from the client to the server.
2. The MAC key for data sent from the client to the server.
3. The encryption key for data sent from the server to the client.
4. The MAC key for data sent from the server to the client.

These keys are derived from the **key derivation function (KDF)**, which takes the master secret and possibly some additional random data to create the keys.

### Data Records

*Why not encrypt data in a constant stream as we write it to TCP?*

Where would we put the MAC? If at the end, there will be no message integrity until all of the data is processed.

Instead, break the stream into a **series of records**. Each record carries a MAC.

The receiver can act on each record as it arrives. We need to be able to distinguish the MAC from the data, as well as using variable-length records.

### Control Information

**Problem**: An attacker can capture and replay, record or reorder records.

**Solution**: Put a **sequence number** in the MAC:

**Problem**: An attacker could replay all the records.

**Solution**: Use a **nonce**.

**Problem**: **Truncation attack** - An attacker forges the TCP connection close segment. One or both sides think that there is less data than there actually is.

**Solution**: Introduce **record types**, with one type for **closure**:

### Outstanding Issues

**How long are the fields?**

**Which encryption protocols are used?**

**Do we want negotiation?**

This allows the client and server to support **different encryption algorithms**. The client and server could choose together a specific algorithm before data transfer.

### SSL Cipher Suite

A cipher suite supplies a:

* Public-key algorithm - RSA
* Symmetric encryption algorithm - DES / 3DES / RC2 / RC4
* MAC algorithm

SSL supports several different cipher suites.

The client and server agree on a cipher suite - client offers a choice from which the server picks.

# Real SSL

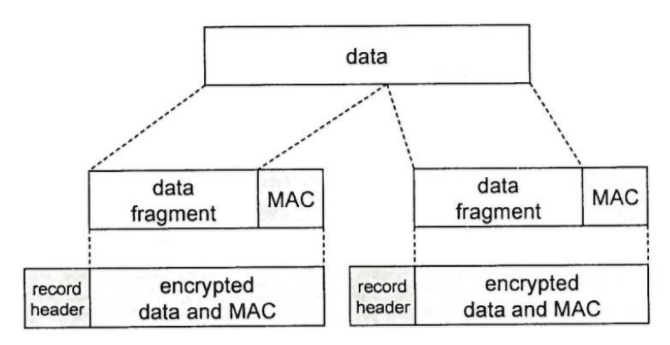
The handshake:

1. The client sends:
   1. List of algorithms it supports
   2. Client nonce.
2. The server sends back:
   1. Choice of algorithms
   2. Certificate
   3. Server nonce
3. The client:
   1. Verifies the certificate
   2. Extracts the server’s public key
   3. Generates pre\_master\_secret
   4. Encrypts pre\_master\_secret with the server’s public key
   5. Sends the encrypted master secret to the server
4. The client and server independently **compute the encryption keys** and **MAC keys** from pre\_master\_secret and the nonces.
5. The client sends a MAC of all the handshake messages.
6. The server sends a MAC of all the handshake messages.

This is all done to make sure that no one has changed the **sequence of events**.

The final two steps prevent a MITM attack from **deleting stronger algorithms** from the list.

### SSL Record Protocol



|  |  |  |
| --- | --- | --- |
| **Record Header**:   * Content type * Version * Length | **MAC**:   * Sequence number * MAC key | **Fragment** = 214 bytes |

### SSL Key Derivation

1. The client nonce, server nonce, and pre-master secret are input into a pseudo-random number generator which produces the **master secret (MS)**.
2. The master secret and nonces are input into another random number generator to produce the **key block**.
3. The key block is sliced into:
   1. Client MAC key
   2. Server MAC key
   3. Client encryption key
   4. Server encryption key
   5. Client initialisation vector (IV)
   6. Server initialisation vector (IV)

# Virtual Private Networks (VPNs)

Institutions often want private networks for **security**, however this is costly.

A VPN allows inter-office traffic to be encrypted before being sent over the internet.

### IPsec

|  |  |
| --- | --- |
| **Transport Mode** | **Tunnel Mode** |
| IPsec datagrams are emitted and received by the **end-systems**. | **End routers** are IPsec aware.  Hosts need not be. |

IPsec offers:

1. **Network-layer authentication**:  
   The destination host can authenticate the source IP addresses.
2. **Network-layer secrecy**:  
   The sending host encrypts the data in an IP datagram.
3. **Two principal protocols**:
   1. **Authentication Header (AH)** protocol
   2. **Encapsulation Security Payload (ESP)** protocol
4. Most common - **Tunnel mode** with **ESP**.

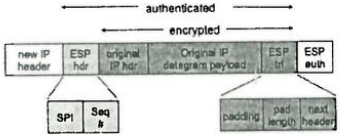


The source and destination perform a **handshake**. This creates a network-layer logical channel called a **Security Association (SA)**. Each SA is unidirectional.

The security association at router 1 has:

1. 32-bit identifier for SA: **Security Parameter Index (SPI)**
2. Origin interface of the SA
3. Destination interface of the SA.
4. Type of encryption used.
5. Encryption key
6. Type of integrity check (e.g. HMAC with MD5)
7. Authentication key

### ESP with Tunnel Mode



1. An **ESP trailer** field is appended to the end of the original datagram.
2. This result is encrypted using the algorithm and key specified by the SA.
3. An **ESP header** is appended to the front of the new encrypted quality.
4. An **authentication MAC** is created over the four fields, using the algorithm and key specified by the SA, and appends the MAC to the **end**.
5. A brand new **IPv4 header** is created and appended before the new payload.

# DNS Security Extension (DNSSEC)

DNS does **not** allow you to check the **authenticity** or **integrity** of a message.

**MITM Attack**:

A resolver has no way to verify the authenticity and integrity of the data sent by name servers.

**Packet Sniffing**:

DNS sends an entire query or response in a single unsigned and unencrypted **UDP packet**.

An attacker can mount an active attack and change the contents.

**Transaction ID Guessing**:

An attacker can respond with false answers to a predicted query.

On the client, there are 232 possible combinations of ID (216) and UDP ports (216).

### DNSSEC

DNSSEC provides origin authenticity and integrity assurance services for DNS data by making use of **digital signatures**.

DNSSEC has two perspectives:

1. **Domain owners** sign their zone and publish the signed zone on their **authoritative name servers**.
2. **Querying hosts** validate the digital signatures they receive in answers along a chain of trust.

DNSSEC adds a number of **new resource record types** such as:

* **RRSIG**: Contains a cryptographic **signature**
* **DNSKEY**: Contains a **public signing key**
* **DS**: Contains a **hash** of a DNSKEY record

### RRsets

The first step towards securing a zone with DNSSEC is to group all records of the same type into a **resource record set (RRset)**. e.g. All AAAA records are bundled into a AAAA RRset.

### Zone-Signing Keys

Each zone in DNSSEC has a **zone-signing key pair (ZSK)**.

A **zone operator** creates digital signatures for **each RRset** using the zone’s **private ZSK**. These are then stored in the name server as **RRSIG records**.

Zone operators also need to make their **public ZSK** available by adding it to their **name server** in a **DNSKEY record**.



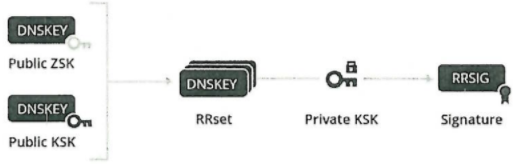
When a DNSSEC resolver requests a particular record type, the name server also returns the corresponding **RRSIG**.

The resolver can then pull the DNSKEY record containing the public ZSK from the name server.

### Key-Signing Keys (KSKs)

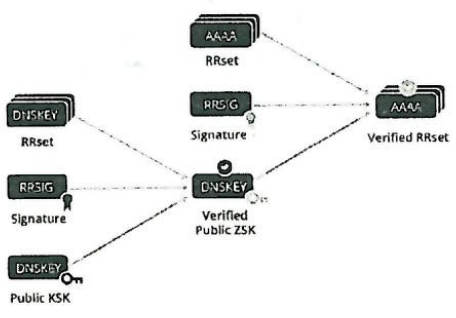
The KSK **validates** the **DNSKEY record**.

It signs the **public ZSK** (which is stored in a DNSKEY record), creating an **RRSIG** for DNSKEY.



### DNSSEC Validation

1. Request the desired **RRset**, which returns the corresponding **RRSIG** record.
2. Request the **DNSKEY** records containing the **public ZSK** and **public KSK**, which returns the **RRSIG** for the **DNSKEY RRset**.
3. Verify the RRSIG of the requested **RRset** with the **public ZSK**.
4. Verify the RRSIG of the **DNSKEY RRset** with the **public KSK**.



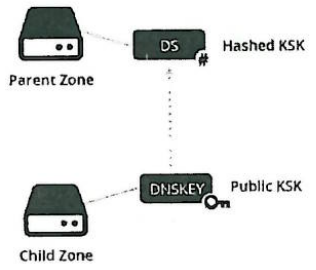
### Delegation Signer Records

DNSSEC introduces a **delegation signer (DS)** record to allow the transfer of trust from a **parent zone** to a **child zone**.

A zone operator **hashes** the **DNSKEY record** containing the **public KSK** and gives it to the **parent zone** to publish as a **DS record**.

Every time a resolver is referred to a child zone, the **parent zone** also provides a DS record.

To check the validity of the child zone’s **public KSK**, the resolver **hashes** it and compares it to the **DS record** from the parent.



04 Electronic Payment Systems

In an online payment, a **third party** is involved at the time of purchase for the verification of payment and to prevent fraud.

The third party can be a **bottleneck** and cause the transaction to take longer.

### Anonymity vs. Audit Trail

An **audit trail** provides a detailed log of all payments:

* Helps prevent fraud.
* Spending profiles can be built.

**Anonymity** protects the identity of the buyer:

* **Full anonymity**: Identity cannot be linked to the payment (e.g. cash).
* **Limited anonymity**: Collaboration could yield the identity.
* Anonymous to the merchant.
* **Privacy**: Payment details are hidden to outsiders.

### Payment Methods

**Macropayments**: >1$ requires **strong** crypto.

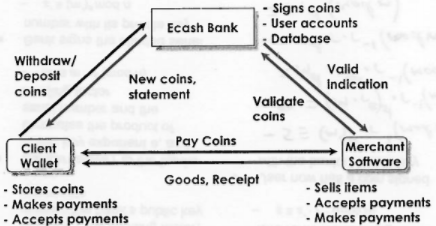
**Micropayments**: <1$ requires **lightweight** crypto.

### eCash

eCash offers **fully anonymous** digital cash:

* Pieces of data representing real monetary value.
* **Digitally signed** by the bank.
* **Problems**:
  + Double spending
  + Exact change

Strong **security** and good **privacy**.



### Blind Signature Protocol

Let:

* be the coin’s serial number
* the blinding factor
* and the bank’s public key exponents

**Protocol**:

1. The sender raises to the bank’s public key exponent and computes the product of the serial number and the blinding factor:
2. The bank signs the blinded serial number with its private key:
3. Returns the coin to the user who removes the blinding factor:
4. The user now has a coin signed with the banks private key:  
     
     
    *( and are inverses)*

# Bitcoin

Bitcoin is a **decentralised**, **peer-to-peer** electronic cash system.

Everyone is collectively the bank ⇒ No single organisation is in charge of the currency.

It makes use of a “**proof-of-work**” concept to prevent double spending in the Bitcoin network. Bitcoin miners are rewarded for **solving** the proof-of-work problem with newly-minted bitcoins or transaction fees.

### Version 1

Alice wants to give Bob a Bitcoin:

1. Alice writes “I, Alice, am giving Bob one Bitcoin”.
2. Alice **digitally signs** the message using her **private key**.
3. Alice announces the signed string of bits to the entire world.

**Problem**: We need a way of making Bitcoins **unique** to prevent double spending:

* Use a label / **serial number**.
* This requires **online** verification.

### Version 2

Make everyone collectively the bank. Everyone keeps a complete record of which Bitcoins belong to which person. This shared public ledger of all Bitcoin transactions is the **Blockchain**.

Alice wants to give Bob a Bitcoin:

1. Alice **signs** the message “I, Alice, am giving Bob one Bitcoin, with serial number 12345”.
2. Bob uses his copy of the Blockchain to check that the Bitcoin is Alice’s to give.  
   He broadcasts both **Alice’s message** and his **acceptance of the transaction** to the entire network, and everyone **updates their copy** of the Blockchain.

### Version 3

When Alice sends Bob a Bitcoin, Bob should **not** try to verify the transaction **alone**.

Broadcast the transaction to the **entire network** of Bitcoin users, and ask them to determine whether the transaction is legitimate.

Alice can only double spend by taking over the Bitcoin network by creating a billion separate entities which are under her control.

### Proof-of-Work (PoW)

PoW requires a combination of two ideas:

1. Make it **computationally costly** for network users to **validate transactions**.
2. **Reward users** for trying to help validate transactions.

As people on the network hear a message, each adds it to a **queue of pending transactions** that they have been told about, but have not yet been approved.

**Example**:

David has the following queue of pending transactions:

1. I, Tom, am giving Sue one Bitcoin, with serial number 1201174
2. I, Alice, am giving Bob one Bitcoin, with serial number 1234567

David checks his copy of the Blockchain and sees that each transaction is valid. He would like to help out by broadcasting news of that validity to the entire network.

### Hash Collisions

As part of the validation protocol, David is required to solve a hard computational puzzle - the PoW.

David needs to find a **nonce**  such that when we **append**  to the list of transactions and **hash** the combination, the output hash begins with a **long run of zeroes** (k-bit partial collision).

This puzzle can be made more or less difficult by varying the number of zeroes.

If we want the output hash to begin with 10 zeroes, on average we would need to try 1610 = 1012 different values for before finding a suitable nonce.

### Bitcoin Miners

If David finds a suitable nonce , he broadcasts the block of transactions he is approving to the network, together with the value of .

Other participants in the network can verify that is valid, and update their Blockchain to include the new block of transactions.

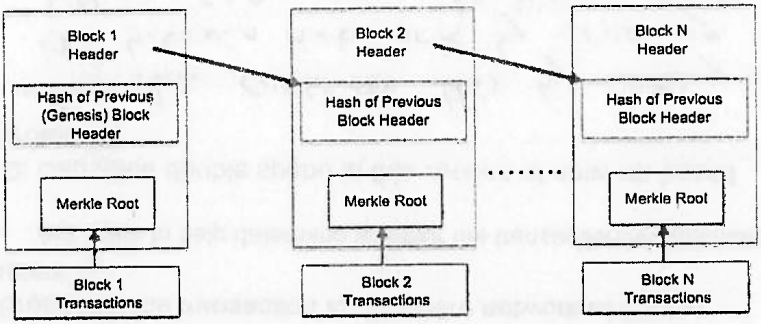
This validation process is called **mining**. For each block of transactions validated, the successful miner receives a bitcoin reward (currently 12.5BTC).

### Bitcoin Transactions



Serial numbers do **not** exist in Bitcoin.

### Blockchain



# Micropayments

Micropayments are **repeated small payments** for low-value information.

**Macropayment problems**:

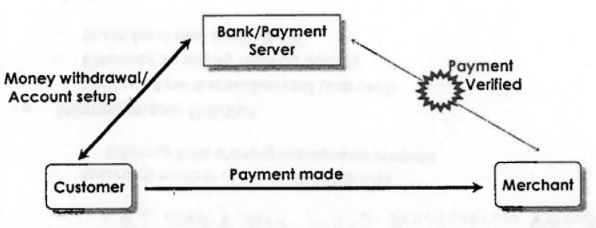
* Minimum price is set by transaction **processing costs**.
* Maximum number of transactions per second - Efficiency limits of strong cryptographic protocols.

**Micropayments solution**:

* Very small per-transaction cost (sub-cent).
* Efficiency by slightly **relaxing security**.
* Some frauds is OK.

Micropayments enable:

* New Internet **opportunities** due to no minimum price for information and services.
* Quality information due to **financial reward**.



Online verification with a third party is **removed**.