Internet Security

- Topics
  - Cryptology (used as tool in many security solutions)
  - TLS
  - DOS attacks
  - DNS security
  - Email security

Security Services

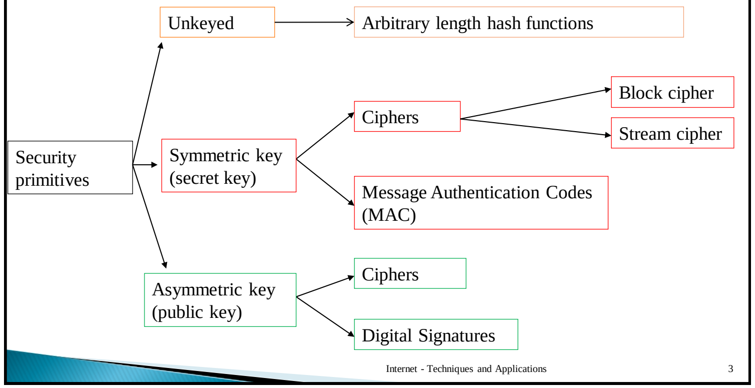
Cryptographic algorithms provide:

- **Confidentiality** – Only authorized users should be able to read a message
- **Integrity** – only authorized users should be able to modify a message
- **Authentication** – we must be able to guarantee that a user is who he claims
- **Nonrepudiation** – a user can not deny having sent or signed a message

- Note that exact definitions vary depending on situation and who you ask

Cryptographic Primitives

Primitives that we will look at

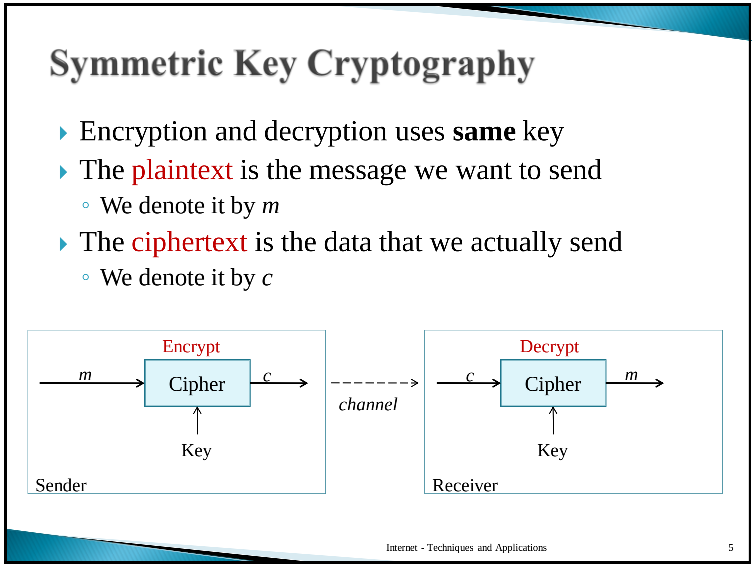

Security primitives
Unkeyed
Arbitrary length hash functions
Symmetric key (secret key)
Ciphers
Block cipher
Stream cipher
Message Authentication Codes (MAC)
Asymmetric key (public key)
Ciphers
Digital Signatures

Strength of Encryption Mechanisms

- Empirically secure – Secure based on the fact that no one has broken it for some time.
  - Most common for practically used symmetric primitives
  - Typically very efficient

- Provably secure – We prove that breaking a scheme is at least as hard as breaking some well known problem like factoring or discrete log.
  - Most common for asymmetric primitives
  - Variants exist for symmetric

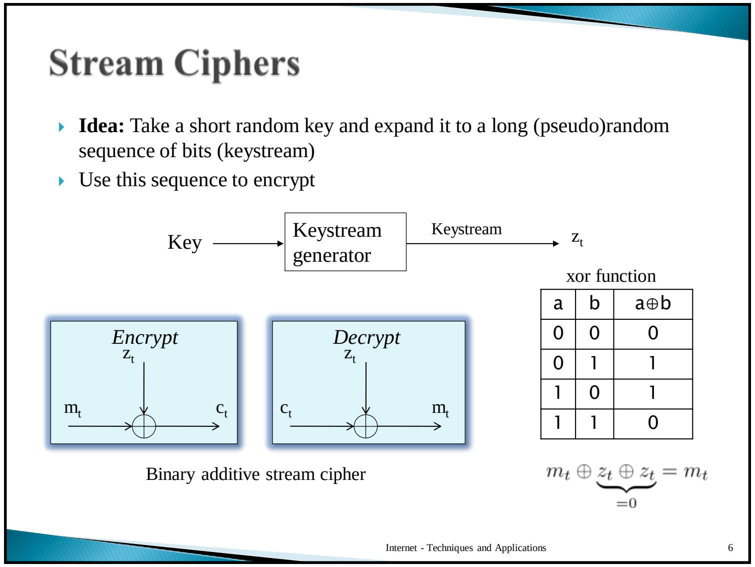
- Unconditionally secure – The schemes are secure even if the adversary has unlimited computing power
  - Not common but possible



Symmetric Key Cryptography

- Encryption and decryption uses **same** key
- The plaintext is the message we want to send
  - We denote it by *m*
- The ciphertext is the data that we actually send
  - We denote it by *c*

Encrypt
m
Cipher
c
Key
Sender
channel
Decrypt
c
Cipher
m
Key
Receiver



Stream Ciphers

- **Idea:** Take a short random key and expand it to a long (pseudo)random sequence of bits (keystream)
- Use this sequence to encrypt

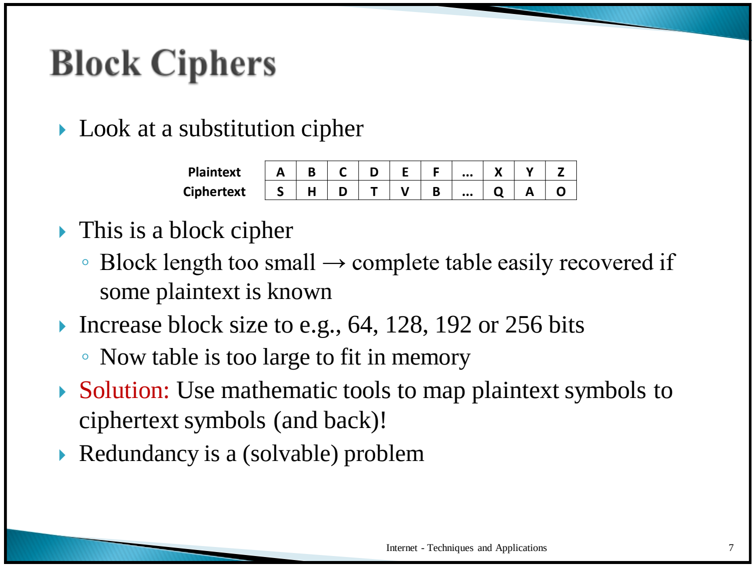
Key
Keystream generator
Keystream
z_t
Encrypt
z_t
m_t
c_t
Decrypt
z_t
c_t
m_t

Binary additive stream cipher

xor function

a	b	a⊕b
0	0	0
0	1	1
1	0	1
1	1	0

$$m_t \oplus \underbrace{z_t \oplus z_t}_{=0} = m_t$$

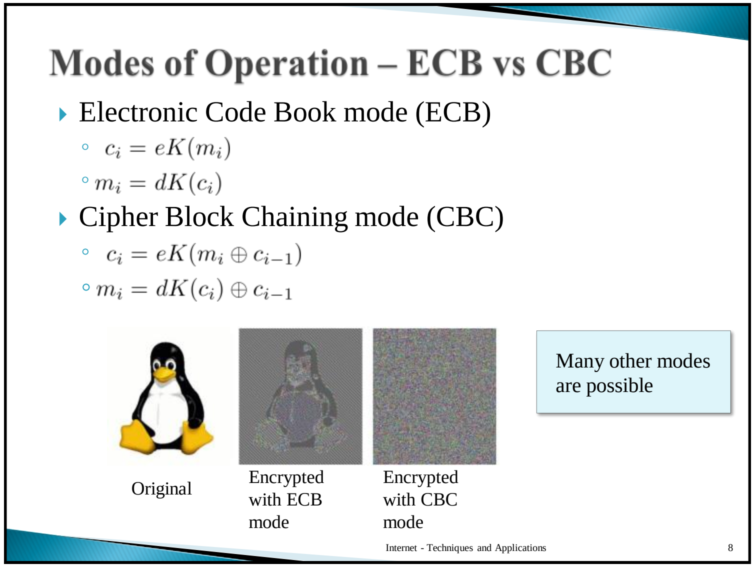


Block Ciphers

- Look at a substitution cipher

Plaintext	A	B	C	D	E	F	...	X	Y	Z
Ciphertext	S	H	D	T	V	B	...	Q	A	O

- This is a block cipher
  - Block length too small → complete table easily recovered if some plaintext is known
- Increase block size to e.g., 64, 128, 192 or 256 bits
  - Now table is too large to fit in memory
- Solution: Use mathematical tools to map plaintext symbols to ciphertext symbols (and back)!
- Redundancy is a (solvable) problem



Modes of Operation – ECB vs CBC

- Electronic Code Book mode (ECB)
  - $c_i = eK(m_i)$
  - $m_i = dK(c_i)$
- Cipher Block Chaining mode (CBC)
  - $c_i = eK(m_i \oplus c_{i-1})$
  - $m_i = dK(c_i) \oplus c_{i-1}$

Original

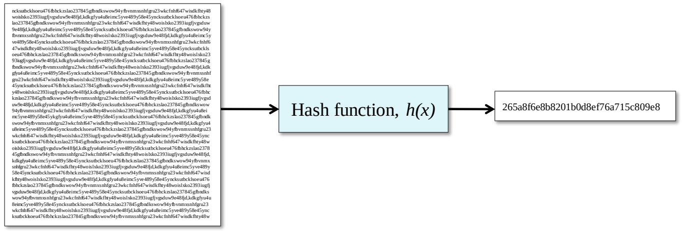
Encrypted with ECB mode

Encrypted with CBC mode

Many other modes are possible

Hash Functions

- Defining properties
  - *Ease of computation:* Easy to compute *h(x)*
  - *Compression: x* of arbitrary bit length maps to fixed length *n* output.

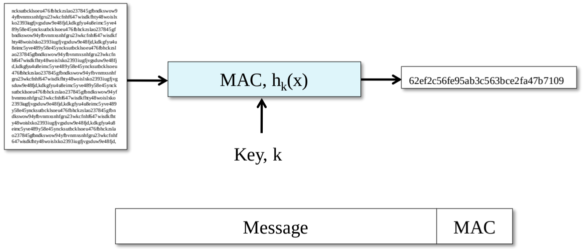


Hash Functions, Properties

- Additional properties
  - *Preimage resistance:* given *y* it is in general infeasible to find *x* such that *h(x)=y*.
  - *Second preimage resistance:* given *x, h(x)* it is infeasible to find *x'* such that *h(x)=h(x')*.
    - Should require around 2^n tries
  - *Collision resistance:* it is infeasible to find *x, x'* such that *h(x)=h(x')*.
    - Should require around $2^{n/2}$ tries, called birthday paradox
- Common hash functions are MD5 and SHA-1 – but they should not be used. SHA-2 and SHA-3 remain secure.

Message Authentication Codes, MACs

- Computed from two inputs, message and a key (*keyed hash functions*)
- *Message authentication codes* proves the integrity of a message (source)



MAC, Properties

- Defining properties
  - *Easy of computation* – Given *k* and *x*, $h_k(x)$ is easy to compute.
  - *Compression* – $h_k(x)$ maps *x* of arbitrary bit length to fixed length *n* output.
  - *Computation resistance* – given zero or more pairs $(x_i, h_k(x_i))$, it is infeasible to compute a pair $(x, h_k(x))$ with a new message *x*.
- *Limitation of MACs*: Transmitter and receiver shares the same key *k*. No possibility to resolve internal disputes.
- Can be constructed using e.g., hash functions or block ciphers

Public Key Cryptography

- Different keys used for encryption and decryption
- Also called *asymmetric cryptography*
  - Public key used to encrypt
  - Private key used to decrypt

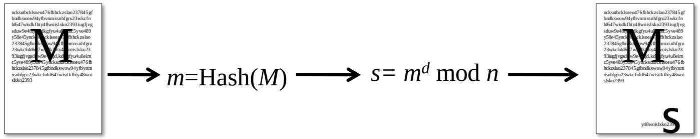
Public Key Cryptography

- Usually based on one of two mathematical problems
  - Factoring – Given an integer n, find the prime factors
  - Discrete Logarithm Problem (DLP) – Given a prime p and integers a and y, find x such that $y = a^x \bmod p$
- RSA is the most well known algorithm
  - Based on factoring
    - $n=pq$, where n is public and (p,q) are private
    - e is public exponent
    - d is private exponent
- Encryption: $c = m^e \bmod n.$
- Decryption: $m= c^d \bmod n$

Informally, **a mod b** is the remainder when **a** is divided by **b**

Digital Signatures

- Private key is used to sign
- Public key is used to verify
- RSA can be used in the same way as encryption
- A **hash value** of the message is signed



$M \longrightarrow m=\text{Hash}(M) \longrightarrow s= m^d \bmod n \longrightarrow$

- Provides nonrepudiation. A MAC does not!
  - A third party can resolve disputes about the validity of a signature without the signer's private key

Comparing Symmetric and Asymmetric Algorithms

- Symmetric algorithms are much faster than asymmetric algorithms. About a factor 1000.
- Symmetric algorithms can use shorter key with same security. 1024 bit RSA modulus corresponds to about 80 bit symmetric key.
- Elliptic curves are often used to make public key cryptography more efficient. Both shorter keys and faster algorithms are possible.

Digital Certificates

public key cryptography:

- Alice has a key pair, one private key and one public key.
- Alice can *sign messages using her private key* and some redundancy in the message (hash value). Anyone can verify the signature using her public key.
- Anyone can *send encrypted messages to Alice using Alice's public key*. Only Alice can decrypt using her private key.
- **Problem:** We need to make sure that the public key we are using really belongs to Alice. Otherwise
  - We may verify a forged signature, thinking it is genuine
  - We may encrypt sensitive data allowing an adversary to decrypt it
- **Solution:** Certificates

Certificates

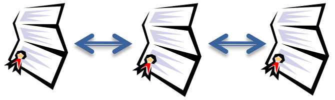
- Primarily binds a subject name to a public key, but can also contain other information such as authorization
- Information is signed by a Certification Authority (CA)
- If CA is trusted, then we trust the binding between user and public key

Public Key Infrastructure

> The set of hardware, software, people, policies and procedures needed to create, manage, store, distribute and revoke digital certificates based on asymmetric cryptography
>
> *RFC 2828, Internet Security Glossary*

Certificate Chains

Verify Alice's public key



Subject: CA
Issuer: CA

Subject: Bob
Issuer: CA

Subject: Alice
Issuer: Bob

1. Receive Alice's certificate containing her name and her public key

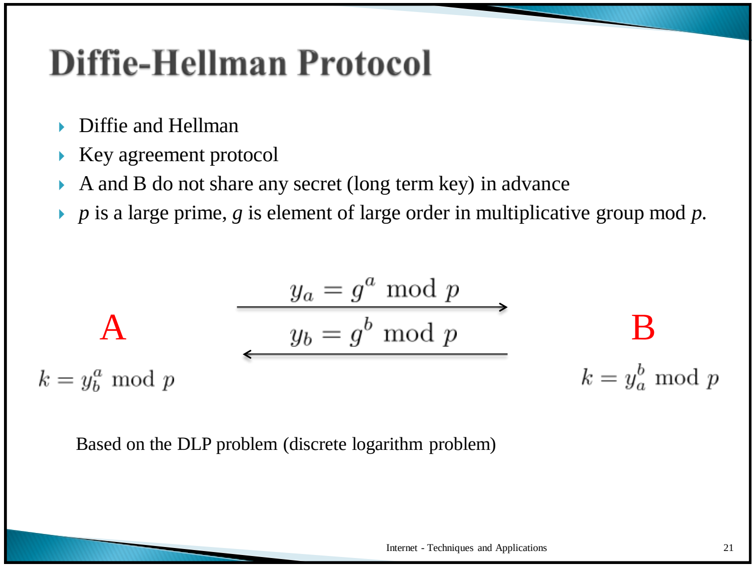
2. We see that it is signed by Bob so we obtain his certificate and verify the signature

3. Bob's certificate is signed with CA's private key so we obtain this certificate and verify the signature

4. The CA certificate is self-signed but if this certificate is among the ones we trust, we decide that the public key of the CA is genuine. We trust Alice's certificate.

Agree on a key

- How would you agree on a key to use with someone else in the room?
- All others are allowed to *listen* to your negotiation
- Is it possible?
- **Hint:** Given a prime p and integers a and y, it is difficult to find x such that $y = a^x \bmod p$



Diffie-Hellman Protocol

- Diffie and Hellman
- Key agreement protocol
- A and B do not share any secret (long term key) in advance
- *p* is a large prime, *g* is element of large order in multiplicative group mod *p*.

A

B

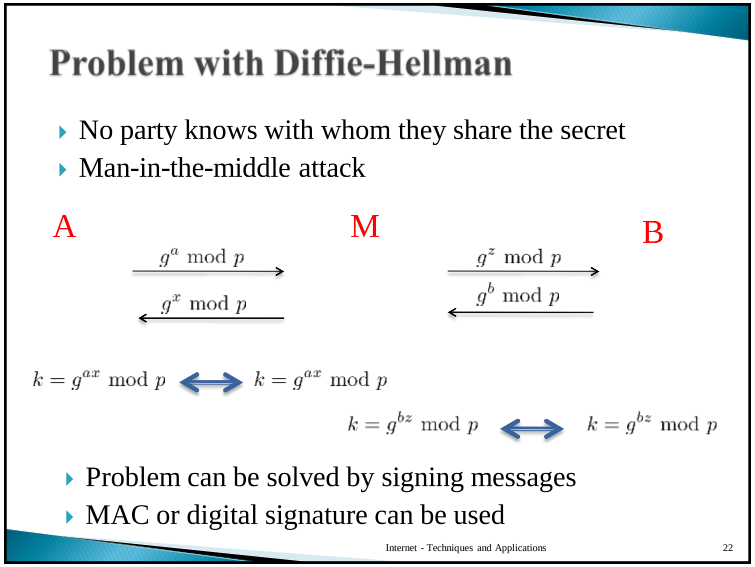
$$y_a = g^a \bmod p \; \longrightarrow$$

$$\longleftarrow \; y_b = g^b \bmod p$$

$k = y_b^a \bmod p$

$k = y_a^b \bmod p$

Based on the DLP problem (discrete logarithm problem)



Problem with Diffie-Hellman

- No party knows with whom they share the secret
- Man-in-the-middle attack

A

M

B

$\xrightarrow{g^a \bmod p}$

$\xleftarrow{g^x \bmod p}$

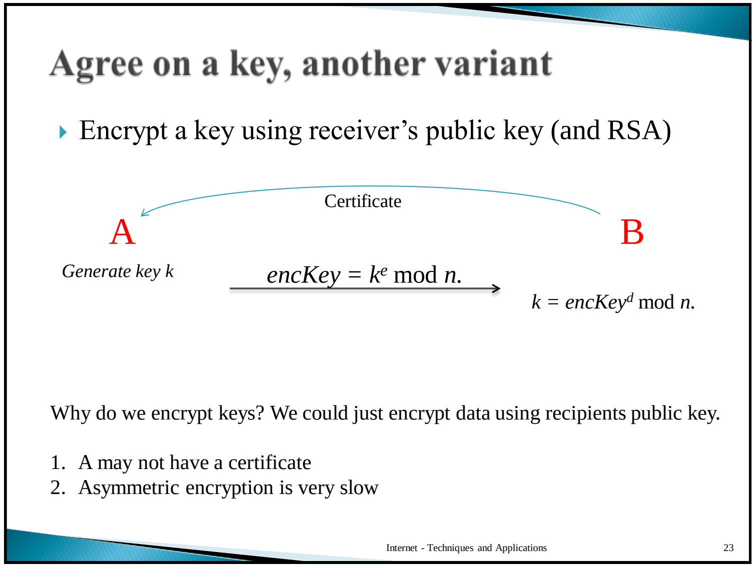
$\xrightarrow{g^z \bmod p}$

$\xleftarrow{g^b \bmod p}$

$k = g^{ax} \bmod p \longleftrightarrow k = g^{ax} \bmod p$

$k = g^{bz} \bmod p \longleftrightarrow k = g^{bz} \bmod p$

- Problem can be solved by signing messages
- MAC or digital signature can be used



Agree on a key, another variant

- Encrypt a key using receiver's public key (and RSA)

Certificate

A

B

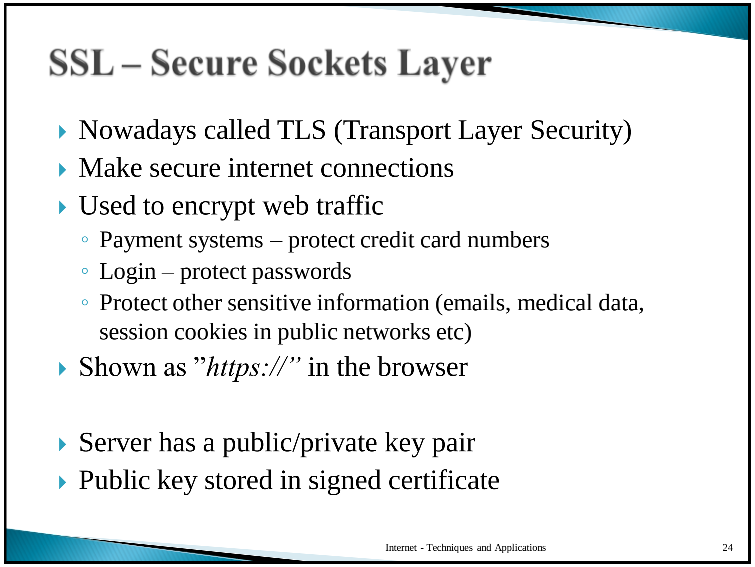
Generate key k

$\xrightarrow{encKey = k^e \bmod n.}$

$k = encKey^d \bmod n.$

Why do we encrypt keys? We could just encrypt data using recipients public key.

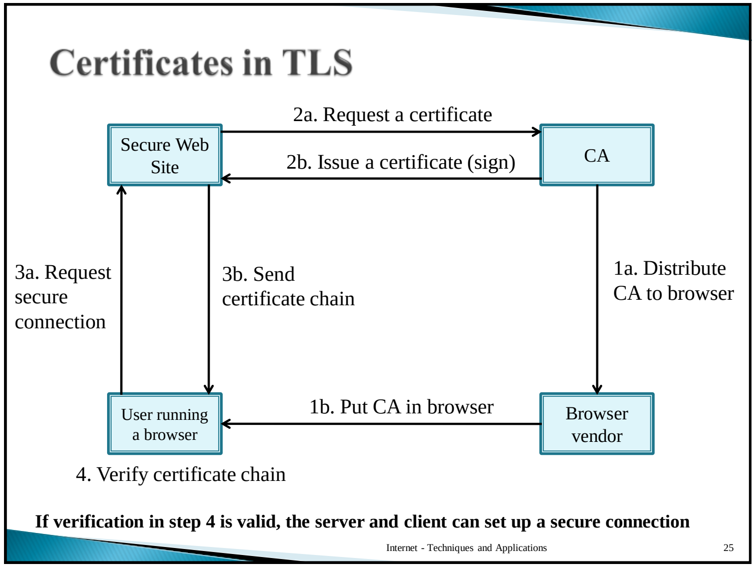
1. A may not have a certificate
2. Asymmetric encryption is very slow



SSL – Secure Sockets Layer

- Nowadays called TLS (Transport Layer Security)
- Make secure internet connections
- Used to encrypt web traffic
  - Payment systems – protect credit card numbers
  - Login – protect passwords
  - Protect other sensitive information (emails, medical data, session cookies in public networks etc)
- Shown as "*https://*" in the browser

- Server has a public/private key pair
- Public key stored in signed certificate



Certificates in TLS

Secure Web Site

2a. Request a certificate

2b. Issue a certificate (sign)

CA

3a. Request secure connection

3b. Send certificate chain

1a. Distribute CA to browser

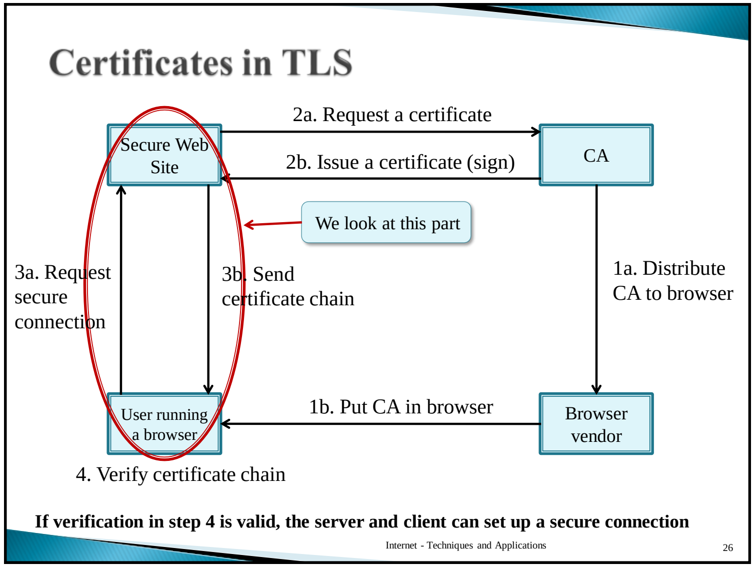
User running a browser

1b. Put CA in browser

Browser vendor

4. Verify certificate chain

If verification in step 4 is valid, the server and client can set up a secure connection



Certificates in TLS

Secure Web Site

2a. Request a certificate

2b. Issue a certificate (sign)

CA

We look at this part

3a. Request secure connection

3b. Send certificate chain

1a. Distribute CA to browser

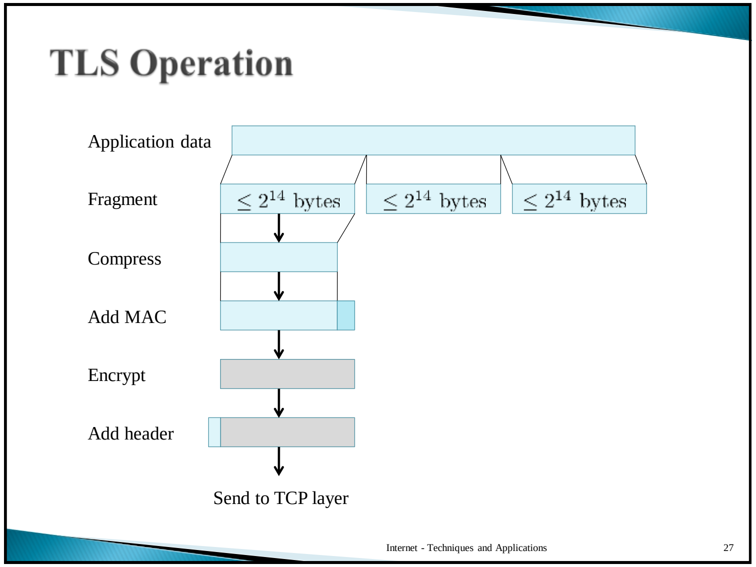
User running a browser

1b. Put CA in browser

Browser vendor

4. Verify certificate chain

If verification in step 4 is valid, the server and client can set up a secure connection



TLS Operation

Application data

Fragment

$\leq 2^{14}$ bytes

$\leq 2^{14}$ bytes

$\leq 2^{14}$ bytes

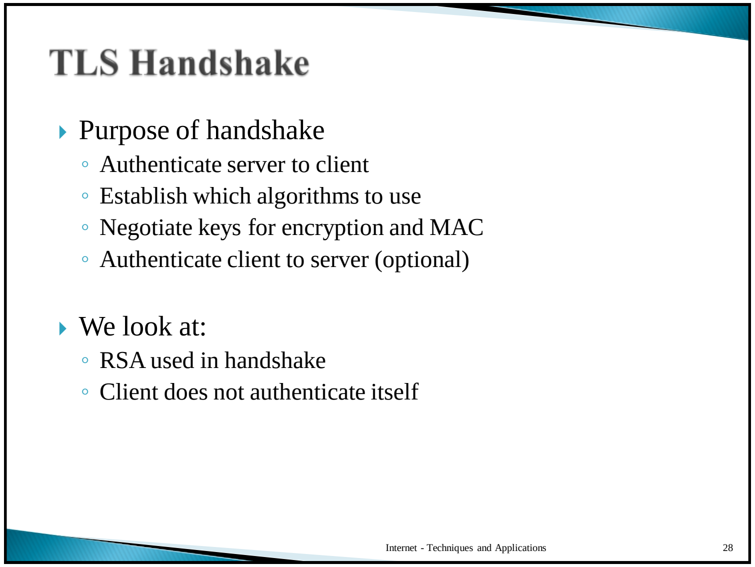
Compress

Add MAC

Encrypt

Add header

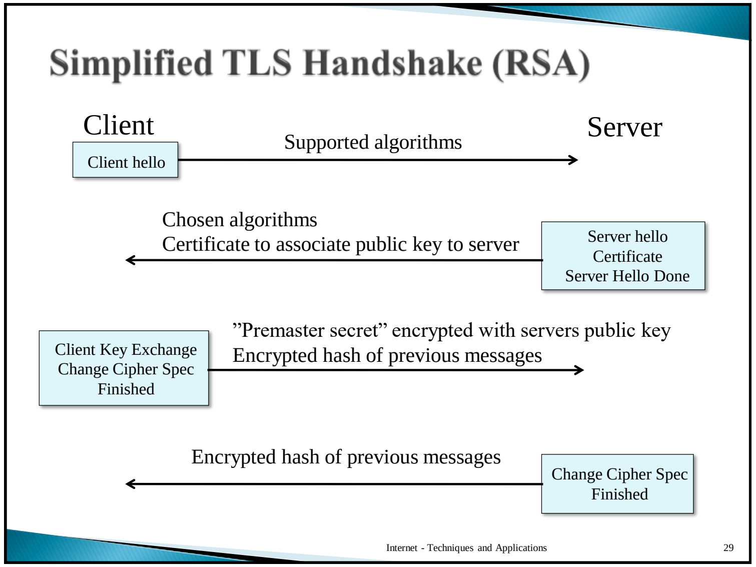
Send to TCP layer



TLS Handshake

- Purpose of handshake
  - Authenticate server to client
  - Establish which algorithms to use
  - Negotiate keys for encryption and MAC
  - Authenticate client to server (optional)

- We look at:
  - RSA used in handshake
  - Client does not authenticate itself



Simplified TLS Handshake (RSA)

Client

Server

Client hello

Supported algorithms

Chosen algorithms
Certificate to associate public key to server

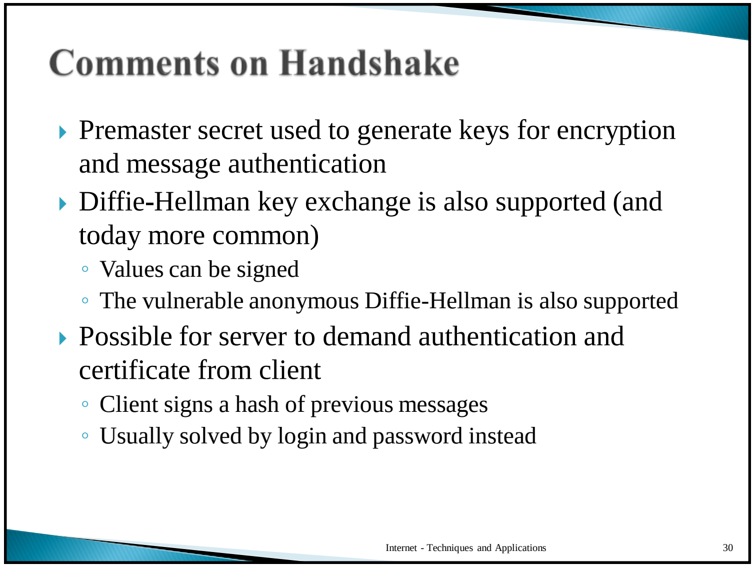
Server hello
Certificate
Server Hello Done

Client Key Exchange
Change Cipher Spec
Finished

”Premaster secret” encrypted with servers public key
Encrypted hash of previous messages

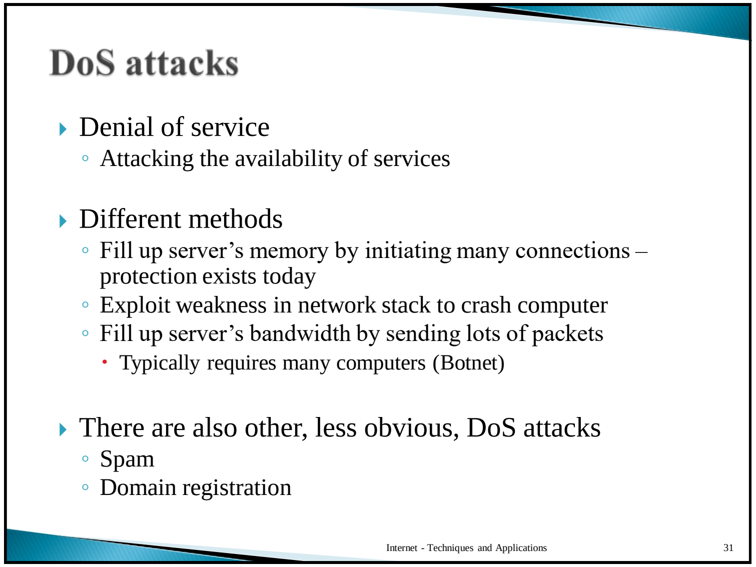
Encrypted hash of previous messages

Change Cipher Spec
Finished



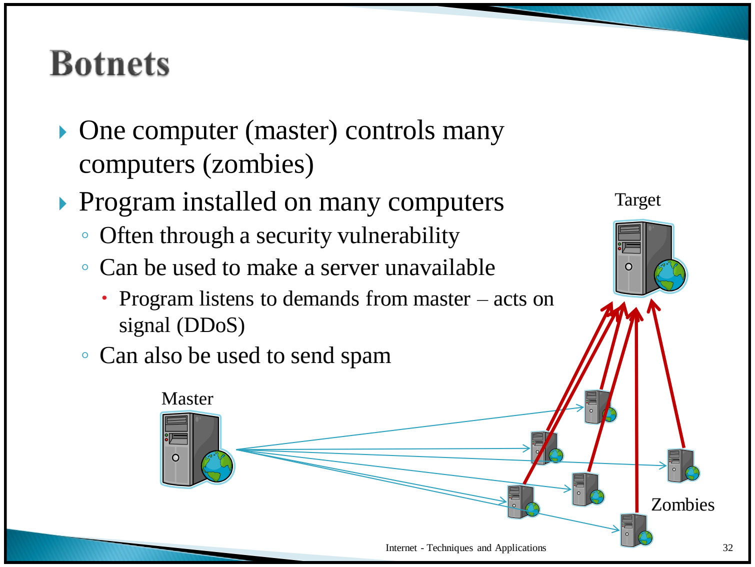
Comments on Handshake

- Premaster secret used to generate keys for encryption and message authentication
- Diffie-Hellman key exchange is also supported (and today more common)
  - Values can be signed
  - The vulnerable anonymous Diffie-Hellman is also supported
- Possible for server to demand authentication and certificate from client
  - Client signs a hash of previous messages
  - Usually solved by login and password instead



DoS attacks

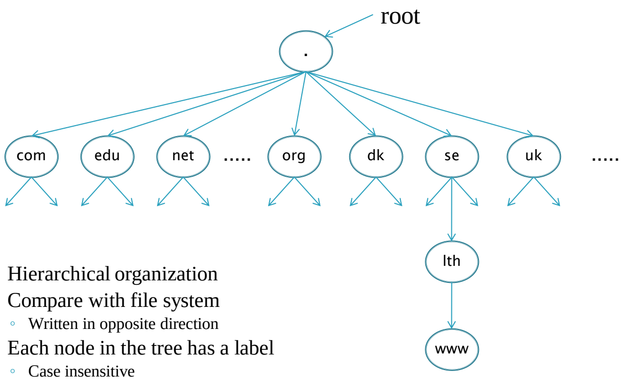
- Denial of service
  - Attacking the availability of services
- Different methods
  - Fill up server’s memory by initiating many connections – protection exists today
  - Exploit weakness in network stack to crash computer
  - Fill up server’s bandwidth by sending lots of packets
    - Typically requires many computers (Botnet)
- There are also other, less obvious, DoS attacks
  - Spam
  - Domain registration



Botnets

- One computer (master) controls many computers (zombies)
- Program installed on many computers
  - Often through a security vulnerability
  - Can be used to make a server unavailable
    - Program listens to demands from master – acts on signal (DDoS)
  - Can also be used to send spam

DNS Tree Structure

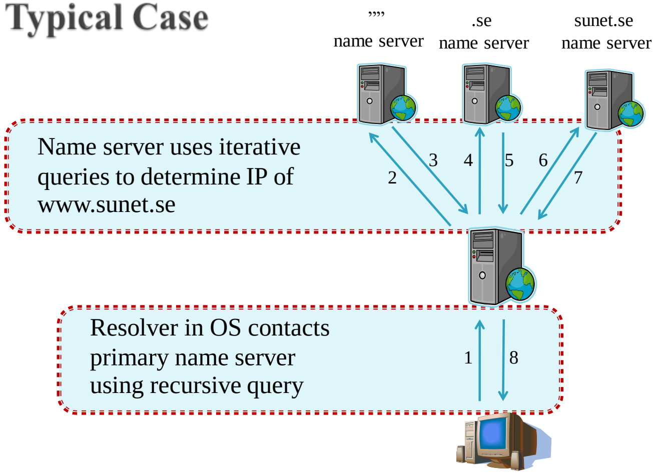


- Hierarchical organization
- Compare with file system
  - Written in opposite direction
- Each node in the tree has a label
  - Case insensitive
- There are 13 root IPs
  - A DNS server must know these IPs

DNS Query

- Parent knows authoritative name servers for its children (but not grand children etc)
- Find IP of server.example.com
  1. Contact a root server to get IP of server.example.com
  2. Responds with IP of name server authoritative of the .com domain
  3. Contact .com name server to get IP of server.example.com
  4. Responds with IP of name server authoritative for example.com
  5. Contact example.com name server to get IP of server.example.com
  6. Responds with IP of server.example.com

Typical Case

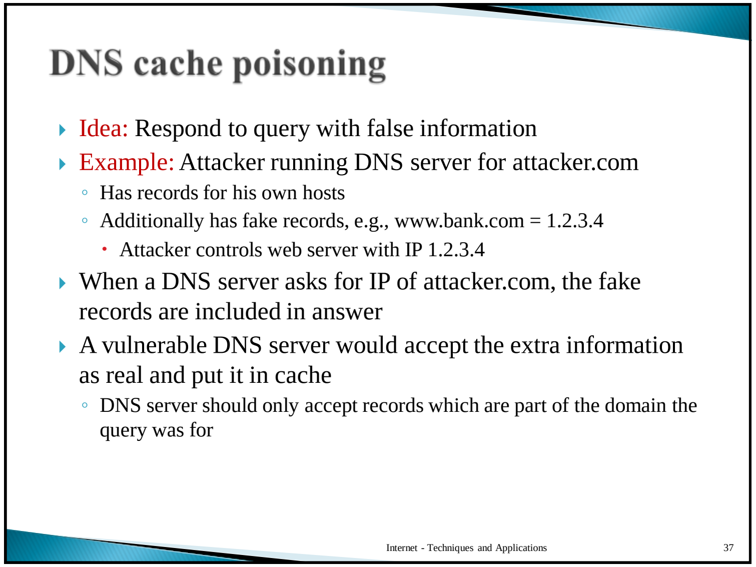


Name server uses iterative queries to determine IP of www.sunet.se

Resolver in OS contacts primary name server using recursive query

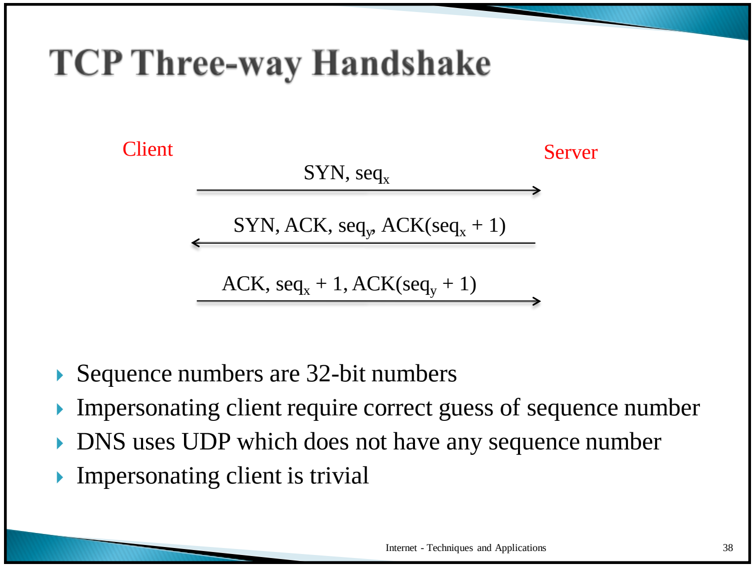
Caching

- Root servers (and TLD servers) would be extremely busy if this was always done
- Instead cache results and reuse them
- If **www.example.com** was first queried and then **www.server.com**, the com name server would be known and root is not contacted.
- TTL value determines for how long records should be cached (order of a few days).



DNS cache poisoning

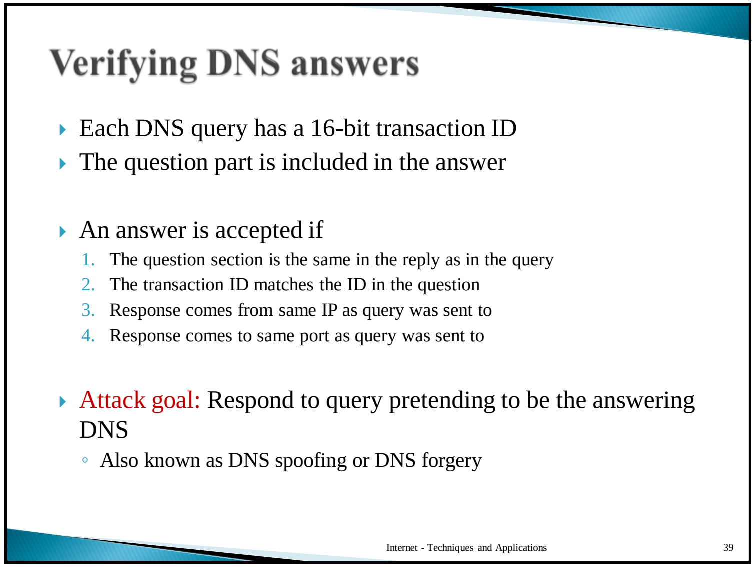
- Idea: Respond to query with false information
- Example: Attacker running DNS server for attacker.com
  - Has records for his own hosts
  - Additionally has fake records, e.g., www.bank.com = 1.2.3.4
    - Attacker controls web server with IP 1.2.3.4
- When a DNS server asks for IP of attacker.com, the fake records are included in answer
- A vulnerable DNS server would accept the extra information as real and put it in cache
  - DNS server should only accept records which are part of the domain the query was for



TCP Three-way Handshake

Client
Server
SYN, seq_x →
← SYN, ACK, seq_y, ACK(seq_x + 1)
ACK, seq_x + 1, ACK(seq_y + 1) →

- Sequence numbers are 32-bit numbers
- Impersonating client require correct guess of sequence number
- DNS uses UDP which does not have any sequence number
- Impersonating client is trivial

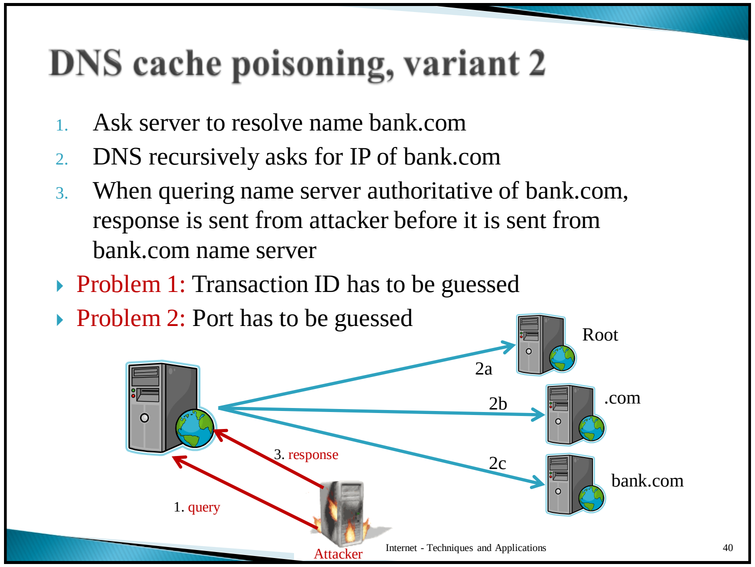


Verifying DNS answers

- Each DNS query has a 16-bit transaction ID
- The question part is included in the answer

- An answer is accepted if
  1. The question section is the same in the reply as in the query
  2. The transaction ID matches the ID in the question
  3. Response comes from same IP as query was sent to
  4. Response comes to same port as query was sent to

- Attack goal: Respond to query pretending to be the answering DNS
  - Also known as DNS spoofing or DNS forgery



DNS cache poisoning, variant 2

1. Ask server to resolve name bank.com
2. DNS recursively asks for IP of bank.com
3. When quering name server authoritative of bank.com, response is sent from attacker before it is sent from bank.com name server

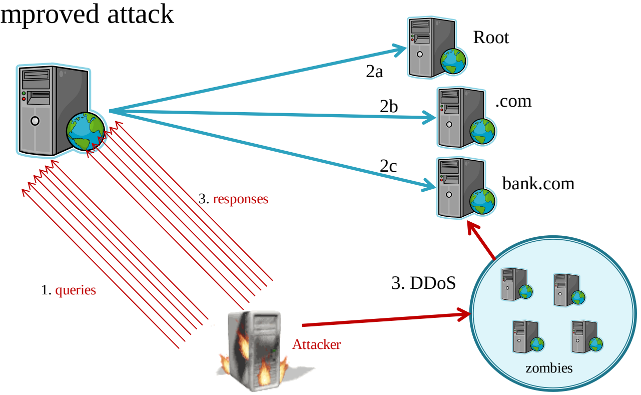
- Problem 1: Transaction ID has to be guessed
- Problem 2: Port has to be guessed

DNS cache poisoning, variant 2

- Port is not always random enough
  - Newer implementations support this better
- Transaction ID is 16 bits
  - We need to send about 65536 responses in order to be lucky with colliding IDs
- Improving the attack:
  - Send several queries at one time
  - If we can guess port then we only need about 300 queries and 300 responses according to ”birthday paradox”
  - All responses must be sent before real response and after query
  - Attacker can buy some time by doing DoS attack on bank.com name server
- If port number is random, the attack is much more difficult

DNS cache poisoning, variant 2

- Improved attack



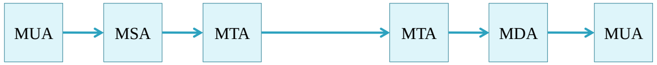
Purpose of DNS cache poisoning

- Identity theft: User enters name, password or other sensitive data on remote site
- Providing false information: Users think they are connected to a site they trust for information
- Man-in-the-middle attacks: After connecting to the attackers site, the site connects to the real site. The attacker is now a man-in-the-middle

- DNSSEC has been proposed as a way to digitally sign the responses
  - Used more and more – but responses are longer

Email Architecture

- Mail User Agent (MUA): email client, provides the user interface
  - Eudora, pine, outlook, kmail, thunderbird, ...
- Mail Submission Agent (MSA)
  - Usually implemented with MTA
- Mail Transfer Agent (MTA): The software used to transfer emails between servers
  - Implements SMTP
  - Sendmail, Microsoft Exchange Server, ...
- Message Delivery Agent (MDA): The software that delivers received email to the MUA
  - procmail
  - Usually implemented with MTA



Some Commands

- HELO – Initiate a mail transaction
- EHLO – Extended HELO
- MAIL FROM: - Provides sender identification
- RCPT TO: - Provides recipient identification
- DATA - Provides message. End of data indicated by ”.” on an empty line

Example, Send a Message

```
S: 220 server.com Ready
C: EHLO client.com
S: 250-server.com greets client.com
S: 250-8BITMIME
S: 250-SIZE
S: 250-DSN
S: 250 HELP
C: MAIL FROM:<sender@client.com>
S: 250 OK
C: RCPT TO:<rec1@server.com>
S: 250 OK
C: RCPT TO:<rec2@server.com>
S: 250 OK
C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
C: This is my message
C: .
S: 250 OK
C: QUIT
S: 221 server.com Service closing transmission channel
```

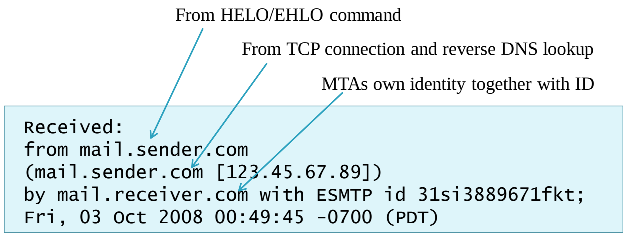
} Extensions supported by server

Mail Headers

- Included in DATA part
- Usually hidden by MUA
- From: and To: header is provided by sender
  - Can easily be forged
- Return-path is added by last SMTP server
  - Used for e.g., error messages if mailbox does not exist
  - Derived from the MAIL FROM command
  - Can have several other names (bounce address, return path, envelope from etc)
- Received: Information added by each involved SMTP server
- Message-id: An ID for the message which is added by the first SMTP server
- X-Header: Headers starting with X- are not part of the standard but used for information only
  - X-Mailer identifies the mailing program
  - X-Headers can be added by anti spam-software, anti-virus software, servers providing web interface, etc

Received Headers Added by Servers

- An MTA must add the header ”received” when it receives/forwards a message
- Example


From HELO/EHLO command
From TCP connection and reverse DNS lookup
MTAs own identity together with ID

```
Received:
from mail.sender.com
(mail.sender.com [123.45.67.89])
by mail.receiver.com with ESMTP id 31si3889671fkt;
Fri, 03 Oct 2008 00:49:45 -0700 (PDT)
```

Sending a Forged Email

- It is easy for anyone to connect to port 25 of an email server and send an email
  - Commands can be chosen arbitrarily
- Without additional checks of involved parties emails can easily be forged
- Headers can be used to track email and (hopefully) find who initiated the email

Example of Forged Email

```
Return-Path: [fake@anywhere.com]
Received: from smtp.server1.com (smtp.server1.com
[134.72.98.54]) by smtp.server2.com with ESMTP id
73659812; Fri, 12 Dec 2007 13:46:54 -0400 (EDT)
Received: from google.com (dklku64.someISP.com
[234.56.67.78]) by smtp.server1.com; Fri, 12 Dec 2007
10:45:28 -0700 (PDT)
Date: Fri, 12 Dec 2007 10:45:28 -0700 (PDT)
From: cheap products <cheap@gmail.com>
To: something@somewhere.org
Subject: The best offer only for you
```

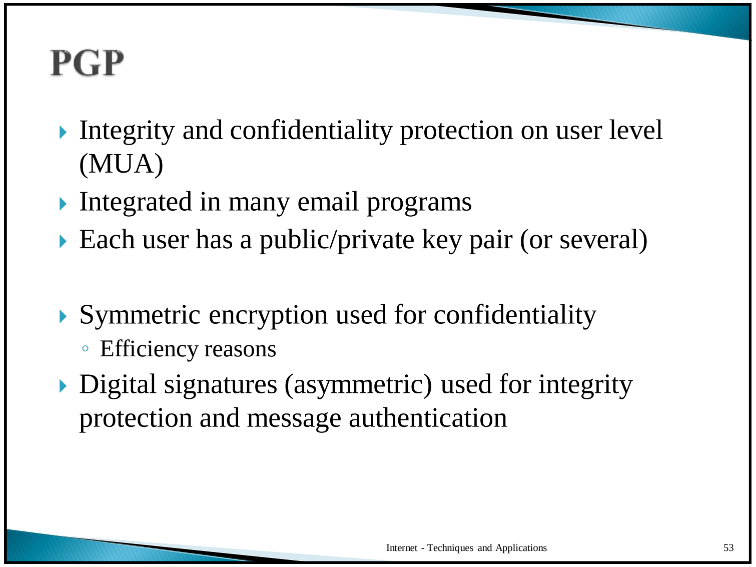
- Here we can see that the bottom received header stems from a forged email
  - Claims that google.com was the SMTP client while it was in fact someone else (234.56.67.78)
- IP used in TCP connection can not be spoofed

DKIM

- DomainKeys Identified Mail, protection at MTA level
- Digital signature of message put in message header
  - Certificates are not used, **public key stored in domain's DNS**
- Allows to verify that the domain has not been spoofed
  - Assuming receiver knows that DKIM should be used for that domain
- Additionally provides integrity protection of message
- Hash algorithm: SHA-256, (SHA-1)
- Signature Algorithm: RSA
- Completely transparent to users

DKIM Example

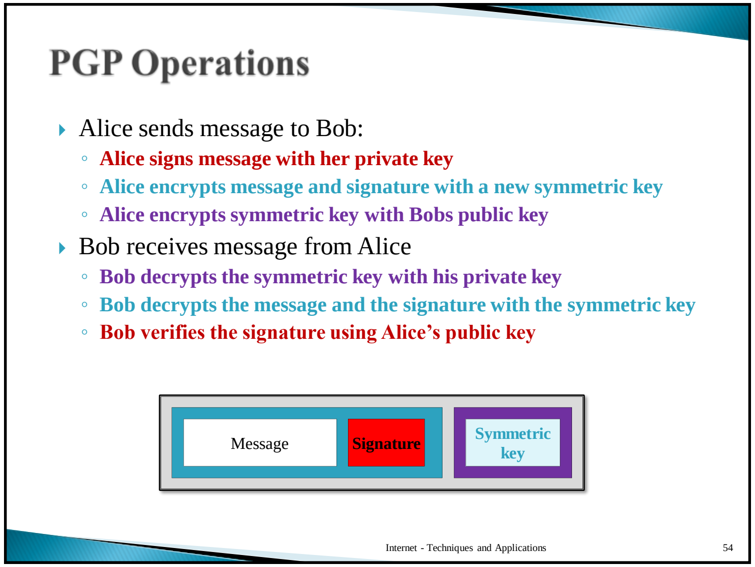
```
DKIM-Signature:
v=1;                Version
a=rsa-sha256;       Algorithms used
c=relaxed/relaxed;Canonicalization (How message was prepared)
d=gmail.com;        Domain
s=gamma;            Selector (to get the correct public key)
h=domainkey-signature:received:received:  Signed headers
message-id:date:from:to:subject:mime-
version:content-type;
bh=9gicsZnlcLK7yYh6VIrgyAMMRZiWsSbWqSPIhc Hash of body
78RRk=;
b=k4ofvpHPkaQmvuSoGVhRrnCsPK+JEuv9KUrZO7a Signature
iypvf/6Y1N2iIatvLvdzwOnZX/W6Kxyx6Z4Ybuk8D
qk/vNTIE7Jpy+GQUUHFvM0NFtmZo1CbGRvo8DdHnX
RBB/qWwlV+Z6wxw/mq7lNuJknVprOAaTLws5mwcZ+
AWL8KwHgO=
```



PGP

- Integrity and confidentiality protection on user level (MUA)
- Integrated in many email programs
- Each user has a public/private key pair (or several)

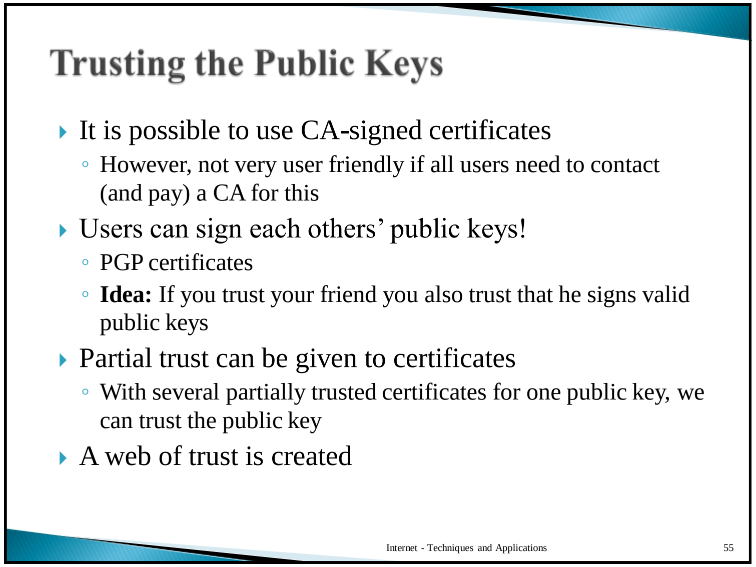
- Symmetric encryption used for confidentiality
  - Efficiency reasons
- Digital signatures (asymmetric) used for integrity protection and message authentication



PGP Operations

- Alice sends message to Bob:
  - **Alice signs message with her private key**
  - **Alice encrypts message and signature with a new symmetric key**
  - **Alice encrypts symmetric key with Bobs public key**
- Bob receives message from Alice
  - **Bob decrypts the symmetric key with his private key**
  - **Bob decrypts the message and the signature with the symmetric key**
  - **Bob verifies the signature using Alice's public key**

Message **Signature** Symmetric key



Trusting the Public Keys

- It is possible to use CA-signed certificates
  - However, not very user friendly if all users need to contact (and pay) a CA for this
- Users can sign each others' public keys!
  - PGP certificates
  - **Idea:** If you trust your friend you also trust that he signs valid public keys
- Partial trust can be given to certificates
  - With several partially trusted certificates for one public key, we can trust the public key
- A web of trust is created