Transport and Application Level Security

Abstract

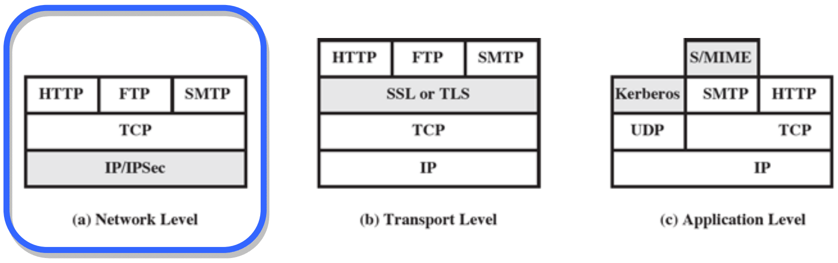
*This section outlines main security protocols at the transport and application level. At the transport level, Secure Sockets Layer (SSL) / Transport Level Security (TLS) is overviewed. At the application level, HTTPS (HTTP over SSL/TLS) for secure web browsing and basics of secure email (*STARTTLS, *S/MIME and PGP) are overviewed.*

Outline

- **Security protocols in the TCP/IP stack**
- Transport Level Security (TLS)
- HTTPS (HTTP over SSL/TLS)
- Secure email

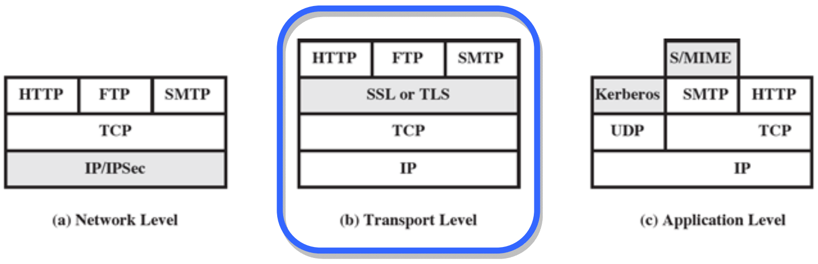
Security Protocols in the TCP/IP Stack

- IP Security (IPsec)
  - transparent to end users and applications
  - general-purpose solution
  - includes filtering so that only selected traffic incurs the overhead of IPsec



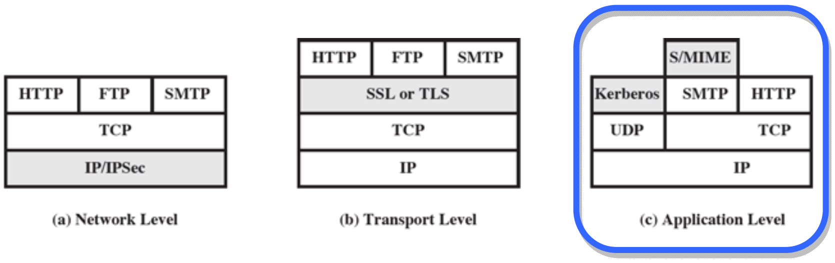
Security Protocols in the TCP/IP Stack

- Secure Sockets Layer (SSL) or Transport Level Security (TLS)
  - could be provided as part of the underlying transport protocol suite (transparent to applications)
  - embedded in specific applications (e.g., web browser)



Security Protocols in the TCP/IP Stack

- Security services embedded within the application
  - each application may implement features to meet its own specific security requirements



Outline

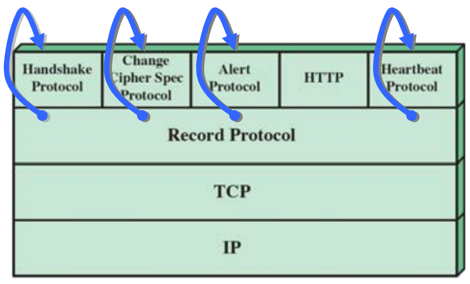
- Security protocols in the TCP/IP stack
- **Transport Level Security (TLS)**
- HTTPS (HTTP over SSL/TLS)
- Secure email

Transport Level Security (TLS)

- One of the most widely used security protocol
- Secure Sockets Layer (SSL) was developed by Netscape for secure HTTP (v1: 1994)
- Transport Level Security (TLS) is an Internet standard (RFC 5246) that evolved subsequently from SSL v3 (1999)
- TLS is a general purpose transport service implemented as a set of protocols that rely on TCP
  - could be provided as part of the underlying transport protocol suite (transparent to applications)
  - embedded in specific applications (e.g., web browser)

SSL/TLC Protocol Stack

- TLS uses basic TCP transport to provide a reliable end-to-end *secure transport service*
- TLS is not a single protocol but rather two layers of protocols
- The TLS Record Protocol provides basic security transport services to higher layer protocols (e.g., HTTP → HTTPS)
- Four higher-layer protocols are defined as part of TLS for the setup or management of TLS connections
  - Handshake Protocol
  - Change Cipher Spec Protocol
  - Alert Protocol
  - Heartbeat Protocol



TLS Connections and Sessions

- TLS *connection*
  - peer-to-peer transport entity providing a secure type of transport service
  - connections are transient
  - every connection is associated with one session
- TLS *session*
  - association between a client and a server
  - created by the Handshake Protocol, with a set of cryptographic security parameters, which can be shared among multiple connections
  - defined to avoid the expensive negotiation of new security parameters for each connection
- Between any pair of parties (e.g., HTTP client and server)
  - there may be multiple secure TLS connections for one session
  - there may also be multiple simultaneous sessions (not used in practice)

TLS Session State

- Session identifier
  - identifies an active or resumable session state
- Peer certificate
  - an X509.v3 certificate of the peer (optional)
- Compression method
  - the algorithm used to compress data prior to encryption
- Cipher spec
  - specifies the bulk data encryption algorithm (null, AES, ...) and a hash algorithm (MD5, SHA-1, …) used for MAC calculation
- Master secret
  - 48-byte secret shared between the client and the server
- Is resumable (flag)

TLS Connection State (1/2)

- Server and client random
  - chosen by the server and client for each connection
- Server write MAC secret
  - secret key used in MAC operations on data sent by the server
- Client write MAC secret
  - secret key used in MAC operations on data sent by the client
- Server write key
  - the symmetric encryption key for data encrypted by the server and decrypted by the client
- Client write key
  - The symmetric encryption key for data encrypted by the client and decrypted by the server

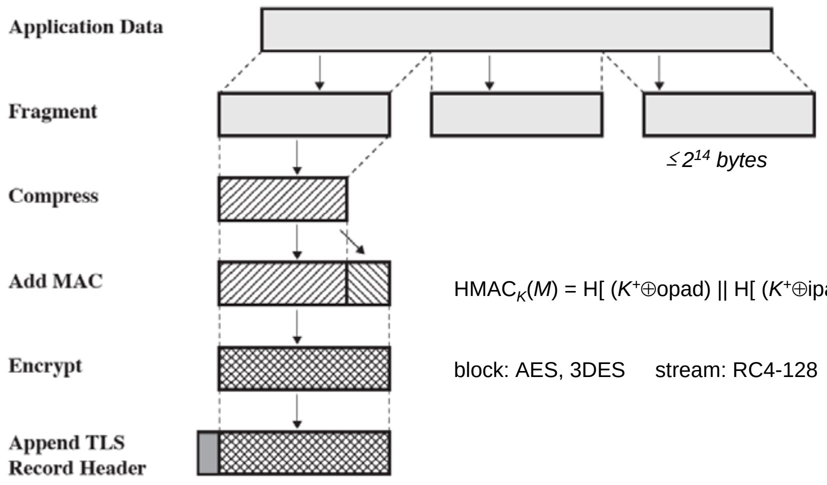
TLS Connection State (2/2)

- Initialization vectors
  - when a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key
  - first initialized by the SSL Handshake Protocol
- Sequence numbers
  - each party maintains separate sequence numbers for transmitted and received messages for each connection
  - when a party sends or receives a change cipher spec message, the sequence number is set to zero

TLS Record Protocol: Services Provided

- Confidentiality
  - the Handshake Protocol defines a shared secret key that is used for conventional *encryption of TLS payloads*
- Message integrity
  - the Handshake Protocol also defines a shared secret key that is used to form a *Message Authentication Code* (MAC)

Operation of the TLS Record Protocol



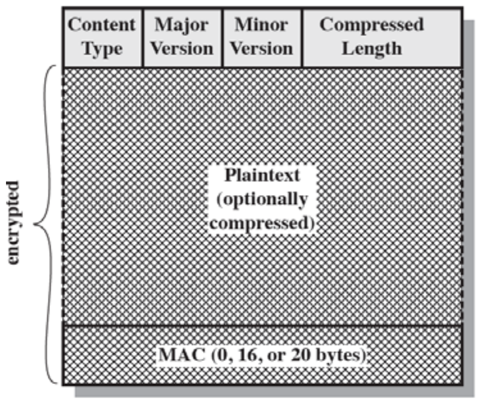
≤ 2^{14} *bytes*

HMAC$_K$(M) = H[(K^+⊕opad) || H[(K^+⊕ipad) || M]]

block: AES, 3DES   stream: RC4-128

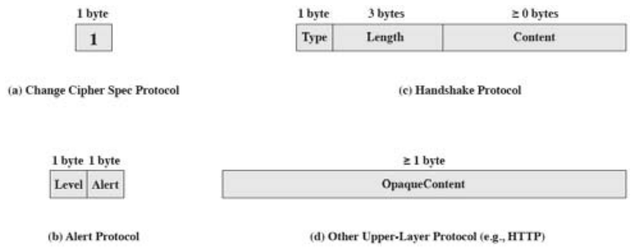
Format of the TLS Record Protocol Message

- *Content Type*: the higher-layer protocol of the enclosed fragment
  - HTTPS
  - Handshake Protocol
  - Change Cipher Spec Protocol
  - Alert Protocol
  - Heartbeat Protocol



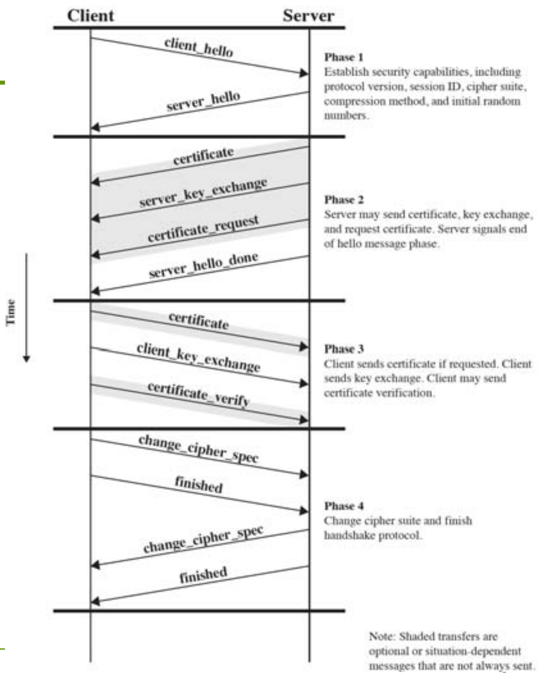
Payload of the TLS Record Protocol Message

- Change Cipher Spec Protocol
  - to update the cipher suite on a connection (pending → current)
- Alert Protocol
  - to convey TLS alerts to the peer entity (closure, warning and fatal alerts)
- Handshake Protocol
  - for server and client authentication
  - to negotiate an encryption and MAC algorithm and cryptographic keys
  - used before any application data is transmitted over TLS Record



Handshake Protocol

- Phase 1:
  establish security functions
  - key exchange method (RSA, DH, …)
  - cipher (RSA, 3DES, …) and MAC algorithms (MD5, SHA)
- Phase 2:
  server authentication and key exchange
- Phase 3:
  client authentication and key exchange
- Phase 4: finish


Client
Server
client_hello
server_hello
Phase 1
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.
certificate
server_key_exchange
certificate_request
server_hello_done
Phase 2
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.
Time
certificate
client_key_exchange
certificate_verify
Phase 3
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.
change_cipher_spec
finished
change_cipher_spec
finished
Phase 4
Change cipher suite and finish handshake protocol.
Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

Handshake Protocol:
Steps to Create Cryptographic Keys (with RSA)

- C→S: random public r_C (32 bytes = TS 4 + random 28) (plain)
- C←S: random public r_S (32 bytes = TS 4 + random 28) (plain)
- C→S: random *pre-master secret* s_{pm} (48 bytes) (encrypted RSA)
- C&S compute the *master secret* (48 bytes) as concatenation of
  - MD5 [s_{pm} || SHA-1($A||s_{pm}||r_C||r_S$)] (128 bit)
  - MD5 [s_{pm} || SHA-1($BB||s_{pm}||r_C||r_S$)] (128 bit)
  - MD5 [s_{pm} || SHA-1($CCC||s_{pm}||r_C||r_S$)] (128 bit)

  A, BB, CCC: padding timestamps counter replay attacks
- From the master secret, C&S compute the *key block*
- From the key block, C&S compute 6 *secret keys*
  - 3 keys for C→S and 3 keys for C←S
    - key 1: data block encryption (3DES, AES)
    - key 2: MAC
    - key 3: IV if CBC mode

Heartbeat Protocol

- *Heartbeat*: periodic signal to indicate normal operation or to synchronize other parts of a system
- Two messages: *heartbeat_request* and *heartbeat_response*
- Two purposes
  - it assures the sender that the recipient is still alive, even though there is no activity over the underlying TCP connection for a while
  - it generates activity across the connection during idle periods (to avoid closure by a firewall that does not tolerate idle connections)

Outline

- Security protocols in the TCP/IP stack
- Transport Level Security (TLS)
- **HTTPS (HTTP over SSL/TLS)**
- Secure email

HTTPS (HTTP over SSL/TLS)

- Combination of HTTP and SSL for secure communication between a web browser and server
  - built in all modern web browsers
  - URL addresses begin with *https://* rather than *http://*
  - If HTTPS is specified, port 443 is used, which invokes SSL
  - documented in RFC 2818, HTTP Over TLS
  - no fundamental change in HTTP over either SSL or TLS
- With HTTPS, the following elements are encrypted
  - URL of the requested document
  - contents of the document
  - contents of browser forms
  - cookies sent from browser to server and from server to browser
  - contents of HTTP header

HTTPS Connection Initiation

- The agent acting as the HTTP client also acts as the TLS client
  - the client begins the TLS handshake
  - once it has finished, the client makes the first HTTP request
  - all HTTP data is sent as TLS application data
- Three levels of connection in HTTPS
  - at HTTP level
    - an HTTP client requests a connection to an HTTP server
    - an HTTP connection request is sent to the next lower layer (TCP or TLS)
  - at TLS level
    - a TLS session is established between a TLS client and a TLS server
    - a TLS session can support one or more TLS connections at any time
    - a TLS request to establish a connection begins with a connection at the lower layer (TCP)
  - at TCP level
    - a TCP connection is established between the TCP entity on the client side and the TCP entity on the server side

HTTPS Connection Closure

- An HTTP client or server can indicate the closing of a connection by including *"Connection: close"*
- Closure of an HTTPS connection involves that
  - → TLS closes the connection with the peer TLS entity
  - → which involves closing the underlying TCP connection
- At the TLS level, the proper way to close a connection is for each side to use the TLS Alert Protocol to send close alerts
- An unannounced TCP closure could be evidence of
  - a programming or communication error
  - some attack

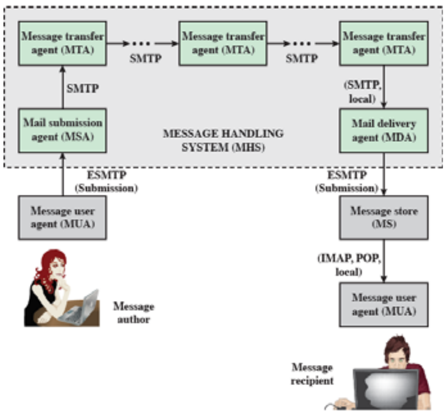
  → the HTTPS client should issue some sort of security warning

Outline

- Security protocols in the TCP/IP stack
- Transport Level Security (TLS)
- HTTPS (HTTP over SSL/TLS)
- **Secure email**

Internet Email Architecture

- RFC 5598, *Internet Mail Architecture*, July 2009
- A user world
  - *Message User Agents* (MUA)
- A transfer world
  - *Message Handling Service* (MHS), composed of *Message Transfer Agents* (MTA), which accepts a message from one user and delivers it to one or more other users

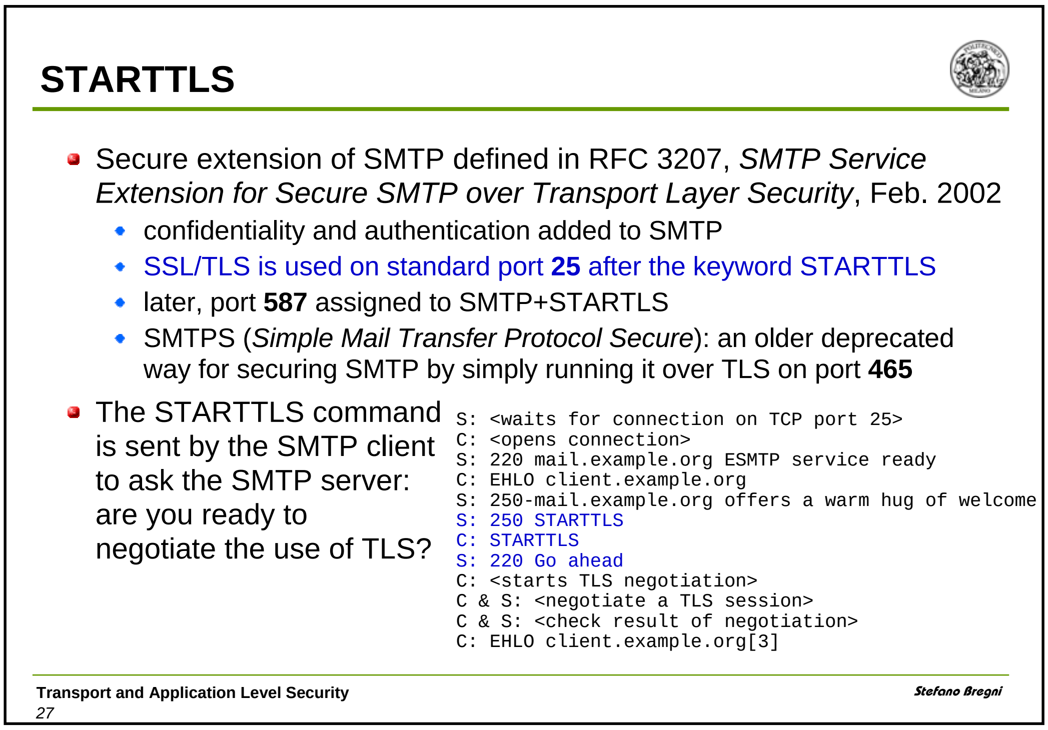


Message Submission and Transfer

Simple Mail Transfer Protocol (SMTP)

- Used to relay the encapsulated messages from source to destination through multiple MTAs
- A text-based client-server protocol where the client (e-mail sender) sends to the server (next-hop recipient) a set of commands
  - the operation of SMTP consists of a series of ASCII commands and responses exchanged between the SMTP sender and receiver
  - a message is transferred to its destination over a SMTP client/server conversation over a single TCP connection or via intermediate relays
  - the SMTP sender establishes the TCP connection (port **25**), then the SMTP conversation begins

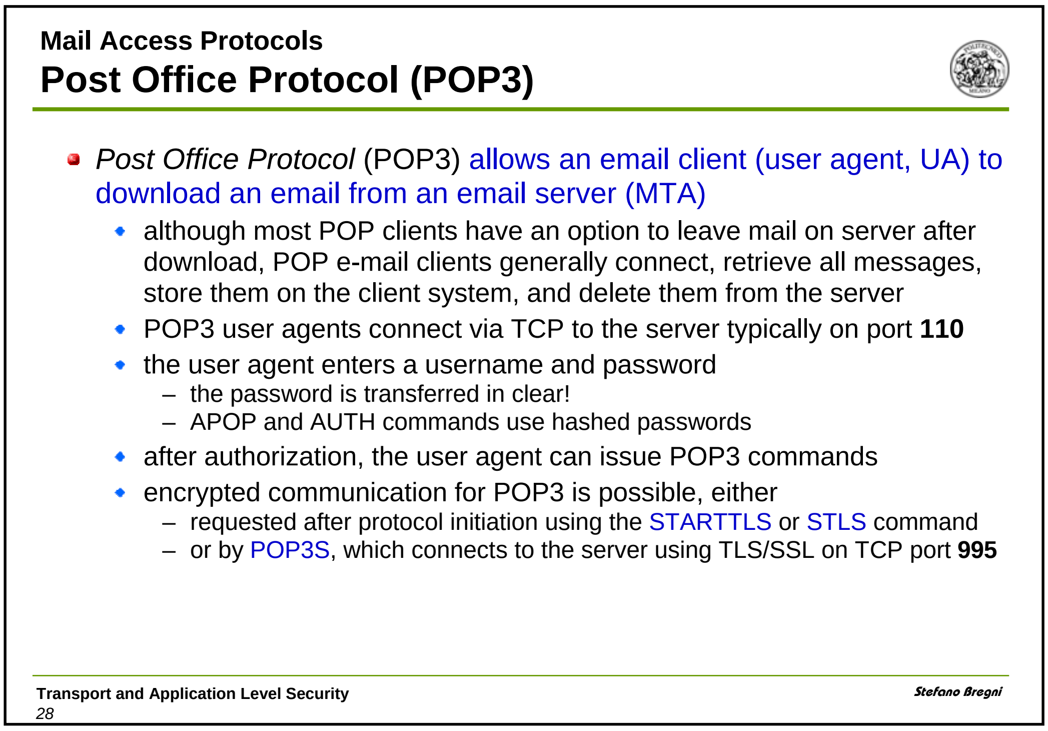
```
S: 220 foo.com Simple Mail Transfer Service Ready
C: HELO bar.com
S: 250 OK
C: MAIL FROM:<Smith@bar.com>
S: 250 OK
C: RCPT TO:<Jones@foo.com>
S: 250 OK
C: RCPT TO:<Green@foo.com>
S: 550 No such user here
C: RCPT TO:<Brown@foo.com>
S: 250 OK
C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
C: Blah blah blah...
C: ...etc. etc. etc.
C: <CRLF><CRLF>
S: 250 OK
C: QUIT
S: 221 foo.com Service closing transmission channel
```



STARTTLS

- Secure extension of SMTP defined in RFC 3207, *SMTP Service Extension for Secure SMTP over Transport Layer Security*, Feb. 2002
  - confidentiality and authentication added to SMTP
  - SSL/TLS is used on standard port **25** after the keyword STARTTLS
  - later, port **587** assigned to SMTP+STARTLS
  - SMTPS (*Simple Mail Transfer Protocol Secure*): an older deprecated way for securing SMTP by simply running it over TLS on port **465**
- The STARTTLS command is sent by the SMTP client to ask the SMTP server: are you ready to negotiate the use of TLS?

```
S: <waits for connection on TCP port 25>
C: <opens connection>
S: 220 mail.example.org ESMTP service ready
C: EHLO client.example.org
S: 250-mail.example.org offers a warm hug of welcome
S: 250 STARTTLS
C: STARTTLS
S: 220 Go ahead
C: <starts TLS negotiation>
C & S: <negotiate a TLS session>
C & S: <check result of negotiation>
C: EHLO client.example.org[3]
```



Mail Access Protocols

Post Office Protocol (POP3)

- *Post Office Protocol* (POP3) allows an email client (user agent, UA) to download an email from an email server (MTA)
  - although most POP clients have an option to leave mail on server after download, POP e-mail clients generally connect, retrieve all messages, store them on the client system, and delete them from the server
  - POP3 user agents connect via TCP to the server typically on port **110**
  - the user agent enters a username and password
    - the password is transferred in clear!
    - APOP and AUTH commands use hashed passwords
  - after authorization, the user agent can issue POP3 commands
  - encrypted communication for POP3 is possible, either
    - requested after protocol initiation using the STARTTLS or STLS command
    - or by POP3S, which connects to the server using TLS/SSL on TCP port **995**

Mail Access Protocols

Internet Mail Access Protocol (IMAP)

- *Internet Mail Access Protocol* (IMAP) also enables an email client to retrieve email on an email server
  - clients generally leave messages on the server until the user explicitly deletes them
  - IMAP user agents connect via TCP to the server typically on port **143**
  - IMAP provides stronger authentication than POP3 and provides more functions not supported by POP3
  - secure IMAP-over-SSL (IMAPS) uses port **993**
  - encrypted communication for IMAP can be requested after protocol initiation using the STARTTLS command

Email Formats

RFC 5322

- Standard format for Internet text email messages
  - messages are viewed as an *envelope* (information needed for transmission and delivery) and *contents* (information to be delivered to the recipient)
- A message consists of some number of header lines (the header) followed by unrestricted text (the body)
  - the header is separated from the body by a blank line
  - a message is ASCII text, where all lines up to the first blank line are assumed to be header lines, used by the user agent of the mail system
  - a header line consists of a keyword followed by a colon and arguments

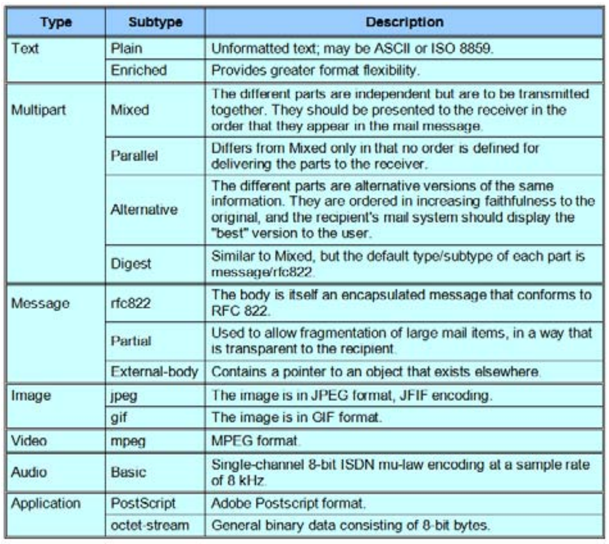
```
Date: Mon, 5 Mar 2018 10:35:24 -0500
To: BREGNI STEFANO <bregni@elet.polimi.it>
From: Sheraton Bogota <reservassheraton@gmail.com>
Reply-To: Sheraton Bogota <reservas@sheratonbogota.com>
Subject: Confirmacion de Reserva
X-Mailer: PHPMailer 5.2.22 (https://github.com/PHPMailer/PHPMailer)

Hello Mr. Bregni,
It is a pleasure to confirm your reservation at Sheraton Bogota Hotel.
```

Email Formats

Multipurpose Internet Mail Extension (MIME)

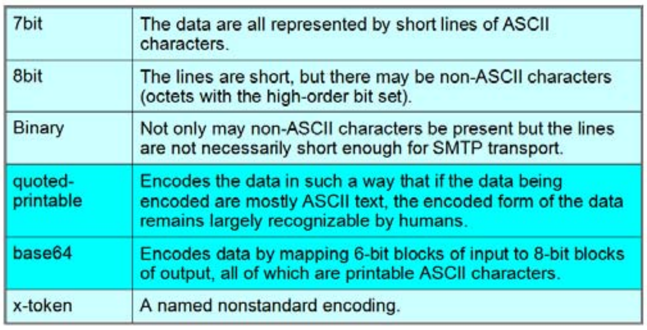
- An extension to RFC 5322 intended to address some of the limitations of SMTP or of RFC 5322
  - SMTP cannot transmit executable or other binary files
    - UNIX UUencode to convert binary files into ASCII form
  - SMTP is limited to 7-bit ASCII
    - cannot transmit text that includes national language characters (8 bits)
    - cannot handle non-textual data included in X.400 messages
  - SMTP gateways that translate between ASCII and EBCDIC do not use a consistent set of mappings
- The MIME specification includes definition of
  - five new message header fields about the body of the message
  - content formats standardizing representations for multimedia email
  - transfer encodings, to convert any content format into a form that is protected from alteration by the mail system

MIME Content Types



Type	Subtype	Description
Text	Plain	Unformatted text; may be ASCII or ISO 8859.
	Enriched	Provides greater format flexibility.
Multipart	Mixed	The different parts are independent but are to be transmitted together. They should be presented to the receiver in the order that they appear in the mail message.
	Parallel	Differs from Mixed only in that no order is defined for delivering the parts to the receiver.
	Alternative	The different parts are alternative versions of the same information. They are ordered in increasing faithfulness to the original, and the recipient's mail system should display the "best" version to the user.
	Digest	Similar to Mixed, but the default type/subtype of each part is message/rfc822.
Message	rfc822	The body is itself an encapsulated message that conforms to RFC 822.
	Partial	Used to allow fragmentation of large mail items, in a way that is transparent to the recipient.
	External-body	Contains a pointer to an object that exists elsewhere.
Image	jpeg	The image is in JPEG format, JFIF encoding.
	gif	The image is in GIF format.
Video	mpeg	MPEG format.
Audio	Basic	Single-channel 8-bit ISDN mu-law encoding at a sample rate of 8 kHz.
Application	PostScript	Adobe Postscript format.
	octet-stream	General binary data consisting of 8-bit bytes.

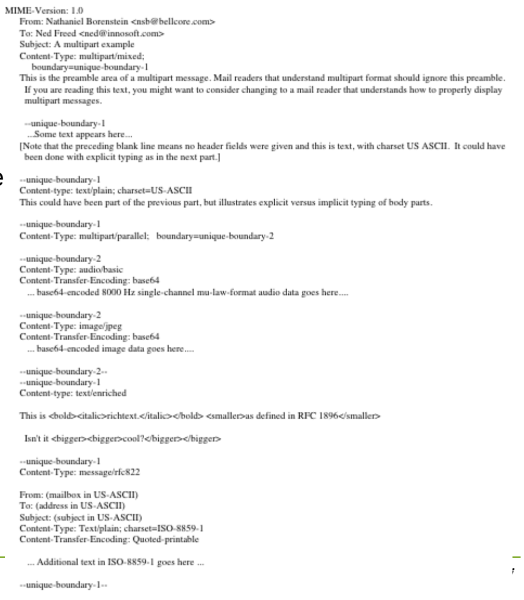
MIME Transfer Encodings



7bit	The data are all represented by short lines of ASCII characters.
8bit	The lines are short, but there may be non-ASCII characters (octets with the high-order bit set).
Binary	Not only may non-ASCII characters be present but the lines are not necessarily short enough for SMTP transport.
quoted-printable	Encodes the data in such a way that if the data being encoded are mostly ASCII text, the encoded form of the data remains largely recognizable by humans.
base64	Encodes data by mapping 6-bit blocks of input to 8-bit blocks of output, all of which are printable ASCII characters.
x-token	A named nonstandard encoding.

Example of MIME Message Structure

- Outline of a multipart MIME message
- Five parts displayed serially
  - two introductory plain text parts
  - an embedded multipart message with two parts to be displayed:
    - a picture fragment
    - an audio fragment
  - a rich-text part
  - a closing encapsulated text message in a non-ASCII character set



```
MIME-Version: 1.0
   From: Nathaniel Borenstein <nsb@bellcore.com>
   To: Ned Freed <ned@innosoft.com>
   Subject: A multipart example
   Content-Type: multipart/mixed;
      boundary=unique-boundary-1
   This is the preamble area of a multipart message. Mail readers that understand multipart format should ignore this preamble.
    If you are reading this text, you might want to consider changing to a mail reader that understands how to properly display
    multipart messages.

    --unique-boundary-1
     ...Some text appears here...
   [Note that the preceding blank line means no header fields were given and this is text, with charset US ASCII.  It could have
    been done with explicit typing as in the next part.]

   --unique-boundary-1
   Content-type: text/plain; charset=US-ASCII
   This could have been part of the previous part, but illustrates explicit versus implicit typing of body parts.

   --unique-boundary-1
   Content-Type: multipart/parallel;   boundary=unique-boundary-2

   --unique-boundary-2
   Content-Type: audio/basic
   Content-Transfer-Encoding: base64
     ... base64-encoded 8000 Hz single-channel mu-law-format audio data goes here....

   --unique-boundary-2
   Content-Type: image/jpeg
   Content-Transfer-Encoding: base64
     ... base64-encoded image data goes here....

   --unique-boundary-2--
   --unique-boundary-1
   Content-type: text/enriched

   This is <bold><italic>richtext.</italic></bold> <smaller>as defined in RFC 1896</smaller>

    Isn't it <bigger><bigger>cool?</bigger></bigger>

   --unique-boundary-1
   Content-Type: message/rfc822

   From: (mailbox in US-ASCII)
   To: (address in US-ASCII)
   Subject: (subject in US-ASCII)
   Content-Type: Text/plain; charset=ISO-8859-1
   Content-Transfer-Encoding: Quoted-printable

     ... Additional text in ISO-8859-1 goes here ...

   --unique-boundary-1--
```

E-mail Security Threats

- Authenticity threats
  - could result in unauthorized access to an e-mail system
- Integrity threats
  - could result in unauthorized modification of e-mail content
- Confidentiality threats
  - could result in unauthorized disclosure of sensitive information
- Availability threats
  - could prevent end users from being able to send or receive e-mail

NIST SP 800-177, *Trustworthy E-mail*, Sep. 2015

Standard Protocols to Counter E-Mail Threats (1)

- STARTTLS
  - SMTP extension that provides authentication, integrity, non-repudiation and confidentiality for the entire SMTP message (SMTP over TLS)
- S/MIME
  - provides authentication, integrity, non-repudiation and confidentiality of the message body carried in SMTP messages
- *DNS Security Extensions* (DNSSEC)
  - authentication and integrity protection of DNS data
  - underlying tool used by various e-mail security protocols
- *DNS-based Authentication of Named Entities* (DANE)
  - provides an alternative channel for authenticating public keys based on DNSSEC
  - the same trust relationships used to certify IP addresses are used to certify servers operating on those addresses

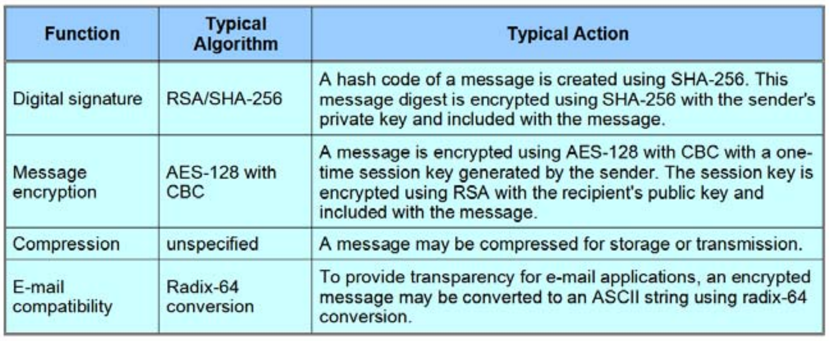
NIST SP 800-177, *Trustworthy E-mail*, Sep. 2015

Standard Protocols to Counter E-Mail Threats (2)

- *Sender Policy Framework* (SPF)
  - uses the DNS to allow domain owners to create records that associate the domain name with a specific IP address range of authorized message senders
  - receivers check DNS to confirm that the purported sender of a message is permitted to use that source address and reject mail that does not come from an authorized IP address
- *Domain Keys Identified Mail* (DKIM)
  - enables an MTA to sign selected headers and the body of a message
  - validates the mail source domain and provides message body integrity
- *Domain-based Message Authentication, Reporting, and Conformance* (DMARC)
  - lets senders know the proportionate effectiveness of their SPF and DKIM policies, and signals to receivers what action should be taken in various individual and bulk attack scenarios

S/MIME

- *Secure / Multipurpose Internet Mail Extension* (S/MIME) is a security enhancement to the standard MIME email format based on RSA
- Provides authentication, integrity, non-repudiation, confidentiality, compression, compatibility of the body carried in SMTP messages



Function	Typical Algorithm	Typical Action
Digital signature	RSA/SHA-256	A hash code of a message is created using SHA-256. This message digest is encrypted using SHA-256 with the sender's private key and included with the message.
Message encryption	AES-128 with CBC	A message is encrypted using AES-128 with CBC with a one-time session key generated by the sender. The session key is encrypted using RSA with the recipient's public key and included with the message.
Compression	unspecified	A message may be compressed for storage or transmission.
E-mail compatibility	Radix-64 conversion	To provide transparency for e-mail applications, an encrypted message may be converted to an ASCII string using radix-64 conversion.

Pretty Good Privacy (PGP)

- Same main functions as S/MIME
  - authentication of the email message
  - encryption of the email message
  - authentication and encryption of the email message
- First released in 1991 by P. Zimmerman as freeware
- The most popular secure email for personal use
- *Web of Trust*: a decentralized system to trust public keys without Certification Authorities
- Current OpenPGP defined in
  - RFC 4880, *OpenPGP Message Format*, 2007
  - RFC 3156, *MIME Security with OpenPGP*, 2001

Main Differences between S/MIME and PGP

- Key Certification
  - S/MIME uses X.509 certificates issued by CAs
  - users of OpenPGP generate their keys and solicit signatures for their public keys from individuals or organizations to which they are known
  - *Web-of-Trust*: an OpenPGP public key is trusted if it is signed by another OpenPGP public key that is trusted by the recipient
- Key Distribution
  - OpenPGP does not include the sender’s public key with each message
  - OpenPGP recipients must obtain the sender’s public key separately
  - organizations post OpenPGP keys on https:// websites
    - who wishes to verify their signatures or send them mail need to download these keys and add them to their OpenPGP client applications (*keyring*)
    - keys may also be registered with OpenPGP public key servers
    - no vetting of keys posted on servers (users rely on the Web-of-Trust)
- NIST recommends S/MIME because of confidence on the CA system

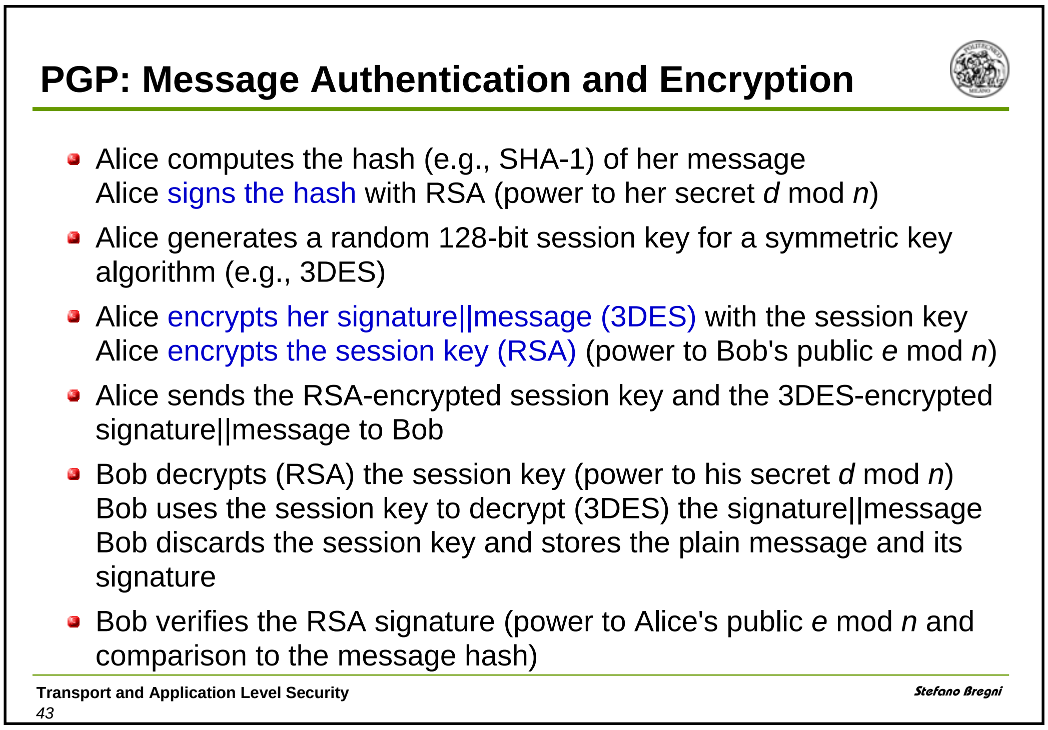
PGP: Message Authentication

- Alice computes the hash (e.g., SHA-1) of her message
- Alice signs the hash with RSA (power to her secret *d* mod *n*)
- Alice transmits the signed hash (signature) to Bob at the beginning of the message
- Bob verifies the RSA signature (power to Alice's public *e* mod *n* and comparison to the message hash)
- *If the RSA signature is verified and Bob trusts Alice's public key*, the message is authenticated and accepted

PGP: Message Encryption

- Alice generates a random 128-bit number to be used as one-time-only session key for a symmetric key encryption algorithm (e.g., 3DES)
  - key generated from random movements of the mouse and keystrokes
- Alice encrypts her message (3DES) with the session key
  Alice encrypts the session key (RSA) (power to Bob's public *e* mod *n*)
- Alice sends the RSA-encrypted key and the 3DES-encrypted message to Bob
- Bob decrypts (RSA) the session key (power to his secret *d* mod *n*)
  Bob uses the session key to decrypt (3DES) the message

- Notes
  - the combination of public-key and symmetric-key algorithms is used because encryption is faster with symmetric key than with public key
  - *trust is not needed, if only encryption is desired*



PGP: Message Authentication and Encryption

- Alice computes the hash (e.g., SHA-1) of her message
  Alice signs the hash with RSA (power to her secret *d* mod *n*)
- Alice generates a random 128-bit session key for a symmetric key algorithm (e.g., 3DES)
- Alice encrypts her signature||message (3DES) with the session key
  Alice encrypts the session key (RSA) (power to Bob's public *e* mod *n*)
- Alice sends the RSA-encrypted session key and the 3DES-encrypted signature||message to Bob
- Bob decrypts (RSA) the session key (power to his secret *d* mod *n*)
  Bob uses the session key to decrypt (3DES) the signature||message
  Bob discards the session key and stores the plain message and its signature
- Bob verifies the RSA signature (power to Alice's public *e* mod *n* and comparison to the message hash)

Transport and Application Level Security
43

Stefano Bregni