Secure Electronic Email

Ibéria Medeiros

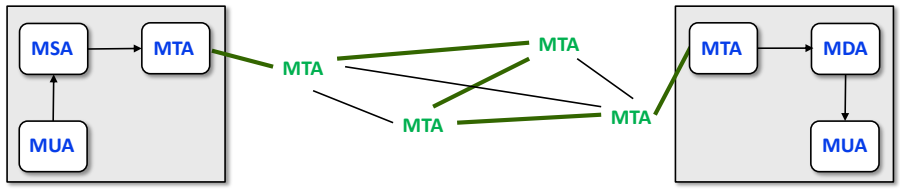
Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa

1

ALTERNATIVES FOR BUILDING *SECURE EMAIL SERVICE*

4

Store-and-Forward Email Model



- Main **components** of the email architecture
  - Mail User Agent (MUA) : allows the end user to write and read of emails from the system; it transfers messages to/from a server; lets the user access a mailbox where the in-bound emails have been stored
  - Mail Transfer Agent (MTA) : forwards the messages towards their destination; communicate using the Simple Mail Transfer Protocol (SMTP)
  - Mail Submission Agents (MSA) : an MTA that accepts emails from MUAs and begins the transmission process by sending it to a MTA; MSA and MTA can be the process
  - Mail Delivery Agent (MDA) : an MTA that receives mail from an organization's inbound MTA and places it in a specific mailbox; MDA and MTA could be the same component

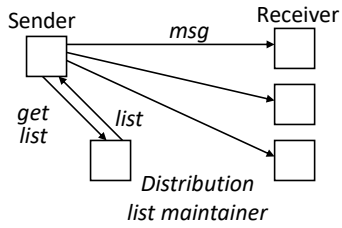
Store-and-Forward Email Model

- **Why do we need MTAs?** Couldn't we just have UAs?
  - some connections may only be available during limited periods of time (e.g., the MTA of the company only connects to the ISP at certain hours of the day)
  - parts of the system might use distinct email protocols (=> need to convert the emails)
  - for security reasons a company might have a *gateway* to process all email traffic

Distribution Lists

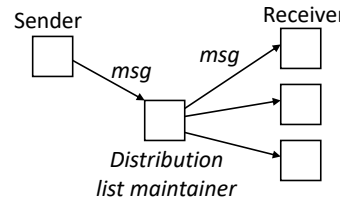
- The user can explicitly indicate the various receivers of an email or can send the email to a ***distribution list***

Local Exploder Method



Some advantages
- easier to prevent distribution loops
- when sending to several lists, it is easier to avoid transmitting duplicate emails to the same receiver

Remote Exploder Method



Some advantages
- keep the confidentiality of the receivers that belong to the list
- more efficient if the list is larger than the email message

Which guarantees would you like to have from the email service?

- **confidentiality:** only the intended receivers should be able to read the email
- **message flow confidentiality:** someone that listens to the network cannot determine if a sender transmitted an email to a certain receiver
- **authentication:** the receiver knows with certainty the identity of the sender
- **anonymity:** the receiver cannot find the identity of the sender
- **integrity:** the email cannot be changed without detection by the receiver
- **message sequence integrity:** ensures that a group of messages is delivered in the same order as they were transmitted, without email losses
- **non-repudiation:** the receiver can later prove to third party that the sender transmitted a message

(... continues in the next slide ...)

Which guarantees would you have from the email service? (cont)

- **proof of submission:** the sender gets from the email service some proof to show that a certain email was transmitted at a given date
- **proof of delivery:** the sender obtains a proof that the receiver got an email at a certain date
- **audit:** the email service records events relevant to the security of exchanged message (e.g., A sent a email to B at 10-Mar-2015)
- **accounting:** keep statistics about the usage of the system
- **self destruct:** the sender can specify that an email should be destroyed after it is read

Let´s start with an easy one, Confidentiality …

- Main points where confidentiality can be broken
  - the messages can be read while being transmitted through the network links
  - MTAs while forwarding the message could keep a copy (which can later be read by an adversary, or by the officials of an organization that wants to monitor the traffic)
- To ensure confidentiality

  How do we get the public key of the receiver?

  How do we do it with only symmetric crypto?
  - the message is encrypted with a symmetric algorithm with a new session key
  - the session key is encrypted with the public key of the receiver

  Advantages: 1 – the message only needs to be encrypted once; 2 – symmetric encryption is faster; 3 – larger lifetime of the public key because it is less used
- Distribution lists
  - [local] *similar to above but with several public keys*
  - [remote] the session key is encrypted with the public key of the maintainer of the list, and then the maintainer decrypts the key and re-encrypts it with the public keys of the list members

    Note: notice that in both cases the sender needs to trust the maintainer!

Authentication of the Sender

- Based on **public key** technology
  - create an hash of the message and encrypt with the private key of the sender
  - how does the receiver get the public key of the sender?
    - » contacts a directory service to obtain a certificate with the public key of the sender
    - » the certificate is transmitted together with the message <== *better*
- Based on **symmetric key** technology (two alternatives)
  - create an hash and encrypt with shared secret key <== *better with several receivers*
  - create an hash of the message concatenated with the secret key
- Distribution lists
  - with public key is simple ....
  - with symmetric key:
    [local] *similar to above, but harder because the sender needs to share a key with receivers*
    [remote] the sender shares a key with the list maintainer, and then the maintainer shares a key with all the receivers

Integrity

- The various mechanisms used to ensure authentication normally also guarantee integrity
- **How can we ensure integrity without the authentication of the sender?**
  - the receiver can determine if there were changes in the message while being transmitted but cannot find out the identity of the sender (e.g., whistleblower or ransom note)

  [symmetric key]
    - » hard (or impossible) because security is based on a shared key

  [public key]
    - » if the message is not sent in the clear, then the email plus an hash can be encrypted with the public key of the receiver
    - » there is always the problem that although the receiver can detect changes to the message, he or she cannot discover if the message was substituted

Non-repudiation (of the Sender)

- The receiver wants to know the identity of the sender, but also wants to be able to prove to a third party that the message came from a certain sender
- Public key technology
  - simple with a message signature
- Symmetric key technology
  - impossible without a *Notary* that is trusted by the users

Method:

1. Alice sends to the Notary an hash of the message (or the message), encrypted with the shared secret key
2. the Notary creates a *seal* for the message (e.g., an hash of the message, the identity of Alice and a secret only known to the Notary)
3. Alice sends the message together with the seal to Bob
4. Bob takes the received message and asks the Notary to verify the seal

Authentication *WITHOUT* Non-repudiation

- The receiver wants to know the identity of the sender, but **cannot** proof to a third party who sent the message (*plausible deniability*)
- Public key technology
  1. Alice generates secret key, *S* (one per message)
  2. Alice encrypts *S* with public key of Bob, $\{S\}_{KU\_Bob}$
  3. Alice signs $\{S\}_{KU\_Bob}$ with her private key, $\{\{S\}_{KU\_Bob}\}SIG_{KR\_Alice}$
  4. Alice uses *S* to calculate a MAC for the message
  5. Alice sends to Bob the MAC, $\{\{S\}_{KU\_Bob}\}SIG_{KR\_Alice}$, and the message
  6. Bob knows that the message was sent by Alice, but cannot prove that a **specific** message was sent because after he receives $\{\{S\}_{KU\_Bob}\}SIG_{KR\_Alice}$ from Alice, he can generate any message and associate a MAC created with *S*
- Symmetric key technology
  - simple because the sender always can say that the receiver created the message (since he or she knows the shared key)

Proof of Submission or Delivery

- Proof of submission
  - simple because the email service (MSA/MTA) could return a receipt (e.g., an hash of the message, date, and other control info, signed by the email service)
- Proof of delivery
  - a receipt is returned after the delivery of the message, signed by the **receiver** or by the **email service**
  - without the cooperation of the receiver it is difficult to achieve this property because he or she may refuse to return the ACK that allows the email service determine the reception of the message
  - if one wants a **proof that the message was read,** then it is even harder
    - » if the receiver signs the receipt before reading the message, then he or she can always say that the signature was made but then a crash occurred that prevented the reading of the message
    - » if the receiver signs the receipt after reading the message, then he or she can refuse to sign

Message Flow Confidentiality and Anonymity

- Message flow confidentiality
  - the adversary cannot find out that Alice sent a message to Bob

  Method 1 : Alice sends the (encrypted) message to a friend that re-transmits it to Bob

  Method 2 : Alice periodically sends (encrypted) messages (valid and garbage) to a set of intermediaries, which then re-send the messages to their intended destination after a random wait period
- Anonymity
  - even if Alice does not sign the message, the receiver can get information about the sender by looking at the header fields added by the MTAs, IP address of sender, etc
  - to ensure anonymity one should use an intermediary that receives messages from many senders

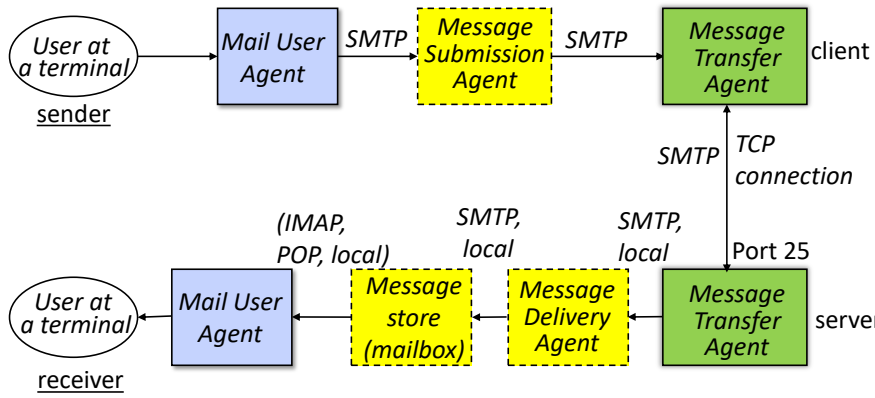
Email System

Email System

- (Some) related standards
  - SMTP - *Simple Mail Transfer Protocol* (RFC 821)
  - definition of message formats (RFC 822, which was later updated to 5322)
  - ESMTP - *Extended Mail Transfer Protocol* (RFC 1424)
  - MIME extensions to the message format (RFCs 2045 to 2049)
  - S/MIME extensions to the message format (RFCs 3370, 3850, 3851, 3852)

Note: There may be more recent versions of these RFCs

SMTP Architecture (According to RFC 5598)



Example user agents: *thunderbird, outlook, mail*
Example message transfer agents: *sendmail, postfix, microsoft exchange*

Overview of the SMTP protocol

1. The client MTA makes a TCP connection to port 25 of the server MTA
2. The server MTA returns its identification and indicates if it is ready to receive messages
3. In the affirmative case, the client MTA indicates the address of the sender and receiver
4. The server MTA responds that the message can be transmitted if the receiver exists
5. The client MTA sends the message and waits for the acknowledgment
6. If the client MTA does not have other messages, it closes the connection

- What to do if the server MTA is not available?
  - the client MTA should save the message in a queue and retry later
  - the initial timeout should be of 30 minutes, and then it should continue to retry periodically until a maximum time is reached (around 3 to 5 days)
  - at that time the message is returned to the sender (the user)
- Format of the commands
  - the commands are ASCII words with 4 letters
  - the responses are typically composed of a numeric code plus some optional textual string (dependent on the MTA implementation and used for debug)

Example: Transmission of an Email

```
telnet mail.di.fc.ul.pt 25
Trying 10.101.86.43...
Connected to mail.di.fc.ul.pt.
Escape character is '^]'.
220 mail.di.fc.ul.pt ESMTP Postfix
>>>HELO caravela.alunos.di.fc.ul.pt
250 mail.di.fc.ul.pt
>>>MAIL From:<i30000@alunos.di.fc.ul.pt>
250 Ok
>>>RCPT To:<nuno@di.fc.ul.pt>
250 Ok
>>>DATA
354 Enter mail, end with "." on a line by itself

Uma mensagem de teste.
.
250 QAA09836 Message accepted for delivery
>>>QUIT
221 mail.di.fc.ul.pt closing connection
Connection closed by foreign host.
```

Use Telnet to connect to the server MTA

Could indicate the supported extensions, including allowing for TLS connections

RFC 822 (or 5322): Text Message Format

- **message envelope:** encapsulates the message and contains all information required for its transmission
- **message contents:** *specified by RFC 822*
  - **header**: control data added by the sender MTA to be used by the receiver agent (e.g., *Return-Path: <i30000@alunos.di.fc.ul.pt>*)
  - **body**: separated from the header with an empty line; contains the data transmitted by the sender; data is composed of ASCII 7-bits characters (most significant bit is 0) and each line should have less than 1000 bytes

```
Return-Path: <i30000@alunos.di.fc.ul.pt>
Received: from caravela.alunos.di.fc.ul.pt (caravela.alunos.di.fc.ul.pt
[10.101.85.31])
        by mail.di.fc.ul.pt (8.8.7/8.8.7) with SMTP id QAA09836
        for <nuno@di.fc.ul.pt>; Thu, 19 Dec 2011 17:16:27 GMT
Date: Thu, 19 Dec 2011 17:16:27 GMT
From: i30000@alunos.di.fc.ul.pt
Message-Id: <199912091656.QAA09836@mail.di.fc.ul.pt>

Uma mensagem de teste.
```

Address from the SMTP command ***mail from***

Address that is seen by the receiver in the **message header** when mail is read

MIME - *Multipurpose Internet Mail Extensions*

- MIME solves some of the problems of RFC 5322 in a way that is compatible with most email systems
  - messages with rich content types (e.g., sound and video)
  - characters other than Latin (e.g., Arab)
  - servers that limit the size of messages or change their content
- MIME solution
  - continue to use the RFC 5322 format, but add structure to messages and ways to encode the data other than ASCII (e.g., 7bit, 8bit, binary, base64, ...)
  - only the user agents need to be changed

New types of header fields

Header	Meaning
MIME-Version:	Identifies the MIME version
Content-Description:	Human-readable string telling what is in the message
Content-Id:	Unique identifier
Content-Transfer-Encoding:	How the body is wrapped for transmission
Content-Type:	Nature of the message

New Types and Subtypes (Content-Type)

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.

Types introduced by S/MIME

multipart/signed

application/pkcs7-signature
application/pkcs7-mime

Encodings (Content-Transfer-Encoding)

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.

Base64

	1 byte						1 byte						1 byte					
Bit pattern	0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0
Index	19						22						5				46							
Base64-Encoded	**T**						**W**						**F**				**u**							

Secure MIME

S/MIME - *Secure MIME*

- Extensions to MIME originally based on technology from RSA to allow, for example, the encryption and signature of emails
- In terms of functionality is relatively equivalent to PGP

- Summary of the functions
  - ***enveloped data*** : data of some type is encrypted and then the session keys are encrypted for one or more receivers
  - ***signed data*** : data is signed by encrypting with the private key an hash of the data; both the signature and data are encoded in base64
  - ***clear-signed data*** : data is signed like in signed-data, but only the signature is encoded in base64 (this means that the data remains readable by agents that only process RFC822 messages)
  - ***signed and enveloped data*** : does both operations above, in a way that *enveloped data* can be signed or *signed data/clear signed-data* can be encrypted

Selection of the Cryptographic Algorithms

- The user transmits in the messages (in the attributes) an indication about which algorithms should be employed to protect future messages that will transmitted to him/her
  - *signing-time attribute* : timestamp when the message was signed
  - *SMIME Capabilities attribute* : list of supported cryptographic algorithms by preference order
  - *Encryption key preference attribute* : indicates which certificate should be used to encrypt the session keys
- The sender chooses the cryptographic algorithms in the following way
  1. if a list of attributes was previously received, then the sender SHOULD employ the algorithms in the preferred order
  2. if no attribute list was received, but some messages have previously arrived from that receiver, then the sender SHOULD employ the same algorithms
  3. if there is no knowledge about the crypto capabilities of the receiver, then the sender SHOULD employ tripleDES because it is more secure; if the sender wants to guarantee the reception of the message then it MUST employ RC2/40

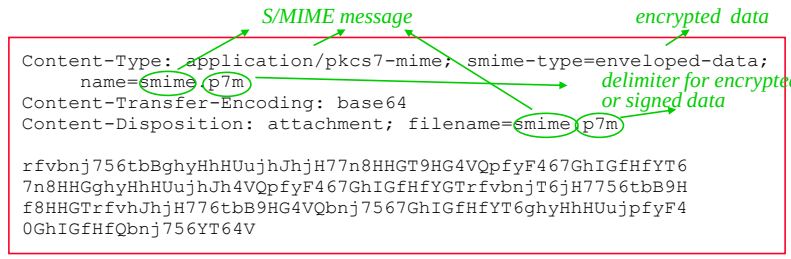
S/MIME New Messages Types

Type	Subtype	smime Parameter	Description
Multipart	Signed		A clear-signed message in two parts: one is the message and the other is the signature.
Application	pkcs7-mime	signedData	A signed S/MIME entity.
	pkcs7-mime	envelopedData	An encrypted S/MIME entity.
	pkcs7-mime	degenerate signedData	An entity containing only public-key certificates.
	pkcs7-mime	CompressedData	A compressed S/MIME entity
	pkcs7-signature	signedData	The content type of the signature subpart of a multipart/signed message.

- Prepare the creation of a new message
  - the data is prepared using the normal MIME methods (called *MIME entity*)
  - S/MIME takes the MIME entity and adds security related information, e.g., algorithm identifiers, to generate a PKCS object (S/MIME entity)
  - the PKCS object is considered like a common MIME data to be transmitted

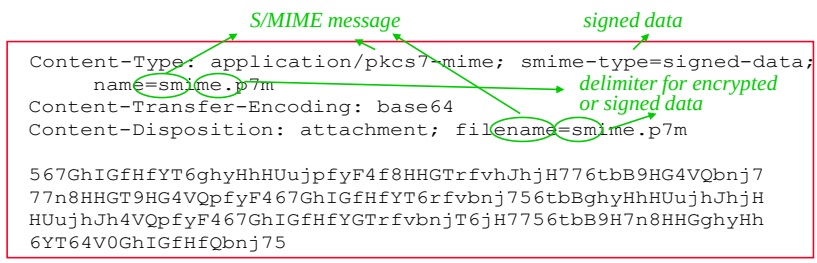
Encrypt a Message – Enveloped data

- Prepare an encrypted message
  - generate an encryption session key (tripleDES, AES or RC2/40)
  - for each receiver, encrypt the session key with the corresponding public key
  - for each receiver, prepare a data block *RecipientInfo* with the identifier of the receiver certificate, the identifier of the encryption algorithm and the encrypted session key
  - encrypt the message data with the session key
  - put together the *RecipientInfo* and encrypted data, and encode with base64



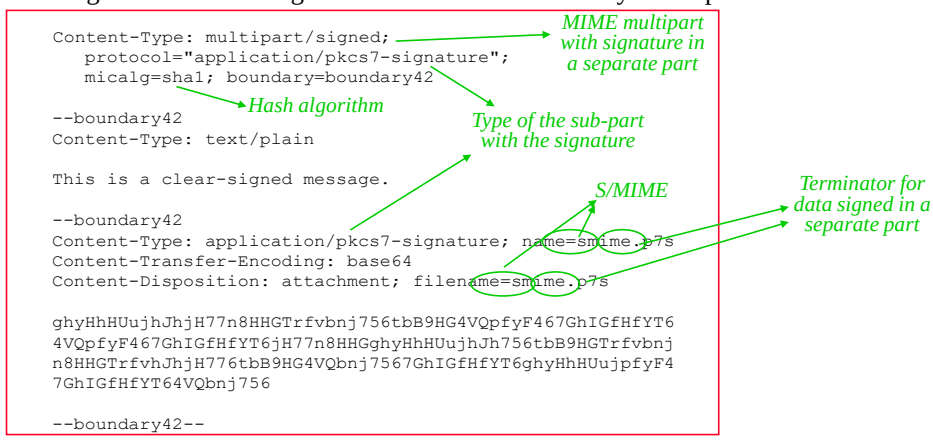
Sign a Message – Signed data

- Prepare a signed message
  - select the hash algorithm (SHA1 or MD5)
  - compute the hash of the data, and the encrypt the hash with the private key
  - prepare a data block *SignerInfo* with the X.509 certificate of the sender, the identifiers of the cryptographic algorithms, and signature
  - put together the *SignerInfo*, the message data and potentially a certificate chain that allows the verification of the sender certificate, and encode in base64



Sign in a Separate Part – Clear signed data

- Consists in using the *multipart* type to send the message data without changes in one part and in the other the signature
- The generation of the signature is done in a similar way as the previous slide



Bibliography

- S. Rose, S. Nightingale, S. Garfinkel, R. Chandramouli, *Trustworthy Email*, NIST Special Publication 800-177, Revision 1, Draft version, December 2017
- W. Stallings, *Cryptography and Network Security (6th Edition)*, 2014 : chapter 19 (pag 619 – 635)
- Kaufman et al, *Network Security: Private Communication in a Public World (2 Edition)*, 2002 : chapter 20 (pag 501 - 525)