ELECTRONIC MAIL SECURITY

Electronic Mail is the most heavily used Network-application and it is also the only distributed application used across all architectures and vendor platforms. Users expect to send mails to others who are connected directly or indirectly to the internet, regardless of host operating systems or communication systems.

With the fast growing reliance on electronic mail for every purpose, there grows a demand for security services such as authentication and confidentiality.

Two schemes, Pretty Good Service (PGP) and S/MIME (Secure/Multipurpose Internet Mail Extension) are used to provide security services to E-mails.

Pretty Good Service (PGP)

PGP is largely the effort of a single person, Phil Zimmermann, PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications.

The properties of PGP are

- PGP is an open-source freely available software package for e-mail security.
- It provides authentication through the use of digital signature, confidentiality through the use of symmetric block encryption.
- It is available free worldwide in versions that run on a variety of platforms, including Windows, UNIX, Macintosh, and many more.
- It is based on algorithms are considered extremely secure. Specifically, the package includes RSA, DSS, and Diffie-Hellman for public-key encryption; CAST-128, IDEA, and 3DES for symmetric encryption; and SHA-1 for hash coding.
- It has a wide range of applicability, from corporations to individuals who wish to communicate securely with others worldwide over the Internet and other networks.
- It was not developed by, nor is it controlled by, any governmental or standards organization.
- PGP is now on an Internet standards track (RFC 3156).
Operational Description.

The actual operation of PGP consists of five services: authentication, confidentiality, Compression, e-mail compatibility, and segmentation as shown in the following Table

<table>
<thead>
<tr>
<th>Sl.No</th>
<th>Function</th>
<th>Algorithms</th>
<th>Used Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Digital signature</td>
<td>DSS/SHA or RSA/SHA</td>
<td>A hash code of a message is created using SHA-1. This message digest is encrypted using DSS or RSA with the sender's private key and included with the message.</td>
</tr>
<tr>
<td>2</td>
<td>Message encryption</td>
<td>CAST or IDEA or Three-key Triple DES with Diffie-Hellman or RSA</td>
<td>A message is encrypted using CAST-128 or IDEA or 3DES with a one-time session key generated by the sender. The session key is encrypted using Diffie-Hellman or RSA with the recipient's public key and included with the message.</td>
</tr>
<tr>
<td>3</td>
<td>Compression</td>
<td>Zip</td>
<td>A message may be compressed, for storage or transmission, using ZIP.</td>
</tr>
<tr>
<td>4</td>
<td>Email compatibility</td>
<td>Radix 64 conversion</td>
<td>To provide transparency for email applications, an encrypted message may be converted to an ASCII string using radix 64 conversion.</td>
</tr>
<tr>
<td>5</td>
<td>Segmentation</td>
<td></td>
<td>To accommodate maximum message size limitations, PGP performs segmentation and reassembly.</td>
</tr>
</tbody>
</table>

Authentication:

Figure 1.1a illustrates the digital signature service provided by PGP and the sequence is as follows:

Digital signatures provide the ability to:

- verify author, date & time of signature
- authenticate message contents
- be verified by third parties to resolve disputes

Hence include authentication function with additional capabilities]

1. The sender creates a message.
2. SHA-1 is used to generate a 160-bit hash code of the message.
3. The hash code is encrypted with RSA using the sender's private key, and the result is prepended to the message.
4. The receiver uses RSA with the sender's public key to decrypt and recover the hash code.
The receiver generates a new hash code for the message and compares it with the decrypted hash code. If the two match, the message is accepted as authentic.

Figure: 1.1 PGP Cryptographic Functions

The combination of SHA-1 and RSA provides an effective digital signature scheme. Because of the strength of RSA, the recipient is assured that only the possessor of the matching private key can generate the signature. Because of the strength of SHA-1, the recipient is assured that no one else could generate a new message that matches the hash code and, hence, the signature of the original message.

Although signatures normally are found attached to the message or file, this is not always the case: Detached signatures are also supported. A detached signature may be stored and transmitted separately from the message it signs.

Detached Signatures are useful in several contexts.

- A user may wish to maintain a separate signature log of all messages sent or received.
- A detached signature of an executable program can detect subsequent virus infection.
A detached signature can be used when more than one party must sign a document, such as a legal contract. Each person's signature is independent and therefore is applied only to the document. Otherwise, signatures would have to be nested, with the second signer signing both the document and the first signature, and so on.

**Confidentiality:**
Confidentiality is provided by encrypting messages to be transmitted or to be stored locally as files. In both cases, the symmetric encryption algorithm CAST-128 (Carlisle Adams and Stafford Tavares) may be used. Alternatively, IDEA (International Data Encryption Algorithm) or 3DES (Data Encryption Standards) may be used. The 64-bit cipher feedback (CFB) mode is used.

As always, one must address the problem of key distribution. In PGP, each symmetric key is used only once. That is, a new key is generated as a random 128-bit number for each message. Thus, although this is referred to in the documentation as a session key, it is in reality a one-time key. Because it is to be used only once, the session key is bound to the message and transmitted with it. To protect the key, it is encrypted with the receiver's public key. Figure 1.1b illustrates the sequence, which can be described as follows:

1. The sender generates a message and a random 128-bit number to be used as a session key for this message only.
2. The message is encrypted, using CAST-128 (or IDEA or 3DES) with the session key.
3. The session key is encrypted with RSA, using the recipient's public key, and is prepended to the message.
4. The receiver uses RSA with its private key to decrypt and recover the session key.
5. The session key is used to decrypt the message.

As an alternative to the use of RSA for key encryption, PGP provides an option referred to as **Diffie-Hellman**

Several observations may be made:
First, to reduce encryption time the combination of symmetric and public-key encryption is used in preference to simply using RSA or ElGamal to encrypt the message directly: CAST-128 and the other symmetric algorithms are substantially faster than RSA or ElGamal.

Second, the use of the public-key algorithm solves the session key distribution problem, because only the recipient is able to recover the session key that is bound to the message. Note that we do not need a session key exchange protocol because we are not beginning an ongoing session. Rather, each message is a one-time independent event with its own key. Furthermore, given the store-and-forward nature of electronic mail, the use of handshaking to assure that both sides have the same session key is not practical.

Finally, the use of one-time symmetric keys strengthens what is already a strong symmetric encryption approach. Only a small amount of plaintext is encrypted with each key, and there is no relationship among the keys. Thus, to the extent that the public-key algorithm is secure, the entire scheme is secure. To this end, PGP provides the user with a range of key size options from 768 to 3072 bits.

**Confidentiality and Authentication:**
As Figure 1.1c illustrates, both services may be used for the same message.
First, a signature is generated for the plaintext message and prepended to the message. Then the plaintext message plus signature is encrypted using CAST-128 (or IDEA or 3DES), and the session key is encrypted using RSA.

In summary, when both services are used, the sender first signs the message with its own private key, then encrypts the message with a session key, and then encrypts the session key with the recipient's public key.

**Compression:**
PGP compresses the message after applying the signature but before encryption. This has the benefit of saving space both for e-mail transmission and for file storage.
The placement of the compression algorithm, indicated by \( Z \) for compression and \( Z^{-1} \) for decompression in Figure 1.1.

1. The signature is generated before compression for two reasons:

   - It is preferable to sign an uncompressed message so that one can store only the uncompressed message together with the signature for future verification. If one signed a compressed document, then it would be necessary either to store a compressed version of the message for later verification or to recompress the message when verification is required.

   - Even if one were willing to generate dynamically a recompressed message for verification, PGP's compression algorithm presents a difficulty. The algorithm is not deterministic; various implementations of the algorithm achieve different tradeoffs in running speed versus compression ratio and, as a result, produce different compressed forms. However, these different compression algorithms are interoperable because any version of the algorithm can correctly decompress the output of any other version. Applying the hash function and signature after compression would constrain all PGP implementations to the same version of the compression algorithm.

2. Message encryption is applied after compression to strengthen cryptographic security. Because the compressed message has less redundancy than the original plaintext, and cryptanalysis is more difficult

**E-mail Compatibility:**

When PGP is used, at least part of the block to be transmitted is encrypted. If only the signature service is used, then the message digest is encrypted (with the sender's private key). If the confidentiality service is used, the message plus signature (if present) are encrypted (With a one-time symmetric key). Thus, part or the entire resulting block consists of a stream of arbitrary 8-bit octets. However, many electronic mail systems only permit the use of blocks consisting of ASCII text. To accommodate this restriction, PGP provides the service
of converting the raw 8-bit binary stream to a stream of printable ASCII characters. The scheme used for this purpose is radix-64 conversion. Each group of three octets of binary data is mapped into four ASCII characters. This format also appends a CRC to detect transmission errors.

The use of radix 64 expands a message by 33%. Fortunately, the session key and signature portions of the message are relatively compact, and the plaintext message has been compressed. In fact, the compression should be more than enough to compensate for the radix-64 expansion. For example, reports an average compression ratio of about 2.0 using ZIP. If we ignore the relatively small signature and key components, the typical overall effect of compression and expansion of a file of length $X$ would be $1.33 \times 0.5 \times X = 0.665 \times X$. Thus, there is still an overall compression of about one-third.

One noteworthy aspect of the radix-64 algorithm is that it blindly converts the input stream to radix-64 format regardless of content, even if the input happens to be ASCII text. Thus, if a message is signed but not encrypted and the conversion is applied to the entire block, the output will be unreadable to the casual observer, which provides a certain level of confidentiality.

Figure 1.2 shows the relationship among the four services so far discussed. On transmission, if it is required, a signature is generated using a hash code of the uncompressed plaintext. Then the plaintext, plus signature if present, is compressed. Next, if confidentiality is required, the block (compressed plaintext or compressed signature plus plaintext) is encrypted and prepended with the public-key-encrypted symmetric encryption key. Finally, the entire block is converted to radix-64 format.

On reception, the incoming block is first converted back from radix-64 format to binary. Then, if the message is encrypted, the recipient recovers the session key and decrypts the message. The resulting block is then decompressed. If the message is signed, the recipient recovers the transmitted hash code and compares it to its own calculation of the hash code.
Segmentation and Reassembly:

E-mail facilities often are restricted to a maximum message length. For example, many of the facilities accessible through the Internet impose a maximum length of 50,000 octets. Any message longer than that must be broken up into smaller segments, each of which is mailed separately.

To accommodate this restriction, PGP automatically subdivides a message that is too large into segments that are small enough to send via e-mail. The segmentation is done after all of the other processing, including the radix-64 conversion. Thus, the session key component and signature component appear only once, at the beginning of the first segment. At the receiving end, PGP must strip off all e-mail headers and reassemble the entire original block.
Cryptographic Keys and Key Rings:

PGP makes use of four types of keys: one-time session symmetric keys, public keys, private keys, and passphrase-based symmetric keys.

Three separate requirements can be identified with respect to these keys:
1. A means of generating unpredictable session keys is needed.
2. We would like to allow a user to have multiple public-key/private-key pairs. One reason is that the user may wish to change his or her key pair from time to time. When this happens, any messages in the pipeline will be constructed with an obsolete key. Furthermore, recipients will know only the old public key until an update reaches them. In addition to the need to change keys over time, a user may wish to have multiple key pairs at a given time to interact with different groups of correspondents or simply to enhance security by limiting the amount of material encrypted with any one key. The upshot of all this is that there is not a one-to-one correspondence between users and their public keys. Thus, some means is needed for identifying particular keys.
3. Each PGP entity must maintain a file of its own public/private key pairs as well as a file of public keys of correspondents.

Session Key Generation:

Each session key is associated with a single message and is used only for the purpose of encrypting and decrypting that message. The message encryption/decryption is done with a symmetric encryption algorithm. CAST-128 and IDEA use 128-bit keys; 3DES uses a 168-bit key.

For the CAST-128, Random 128-bit numbers are generated using CAST-128 itself. The input to the random number generator consists of a 128-bit key and two 64-bit blocks that are treated as plaintext to be encrypted. Using cipher feedback mode, the CAST-128 encryptor produces two 64-bit cipher text blocks, which are concatenated to form the 128-bit session key.

The "plaintext" input to the random number generator, consisting of two 64-bit blocks, is itself derived from a stream of 128-bit randomized numbers. These numbers are based on keystroke input from the user. Both the keystroke timing and the actual keys struck are used
to generate the randomized stream. Thus, if the user hits arbitrary keys at his or her normal pace, a reasonably "random" input will be generated. This random input is also combined with previous session key output from CAST-128 to form the key input to the generator. The result, given the effective scrambling of CAST-128, is to produce a sequence of session keys that is effectively unpredictable.

**Key Identifiers:**

An encrypted message is accompanied by an encrypted form of the session key that was used for message encryption. The session key itself is encrypted with the recipient's public key. Hence, only the recipient will be able to recover the session key and therefore recover the message. If each user employed a single public/private key pair, then the recipient would automatically know which key to use to decrypt the session key: the recipient's unique private key. However, we have stated a requirement that any given user may have multiple public/private key pairs.

How does the recipient know which of its public keys was used to encrypt the session key? One simple solution would be to transmit the public key with the message. The recipient could then verify that this is indeed one of its public keys, and proceed. This scheme would work, but it is unnecessarily wasteful of space. An RSA public key may be hundreds of decimal digits in length.

Another solution would be to associate an identifier with each public key that is unique at least within one user. That is, the combination of user ID and key ID would be sufficient to identify a key uniquely. Then only the much shorter key ID would need to be transmitted. This solution, however, raises a management and overhead problem:

**Key IDs must be assigned and stored so that both sender and recipient could map from key ID to public key.**

The solution adopted by PGP is to assign a key ID to each public key that is, with very high probability, unique within a user ID. The key ID associated with each public key consists of its least significant 64 bits. That is, the key ID of public \( PUa \) is \( (PUa \mod 2^{64}) \). This is a sufficient length that the probability of duplicate key IDs is very small.
A key ID is also required for the PGP digital signature. Because a sender may use one of a number of private keys to encrypt the message digest, the recipient must know which public key is intended for use. Accordingly, the digital signature component of a message includes the 64-bit key ID of the required public key. When the message is received, the recipient verifies that the key ID is for a public key that it knows for that sender and then proceeds to verify the signature.

With the concept of key ID, we can take a more detailed look at the format of a transmitted message, which is shown in Figure 1.3. A message consists of three components: the message component, a signature (optional), and a session key component (optional).

The message component includes the actual data to be stored or transmitted, as well as a filename and a timestamp that specifies the time of creation.

The **signature component** includes the following:

**Timestamp:** The time at which the signature was made.

**Message digest:** The 160-bit SHA-1 digest, encrypted with the sender's private signature key. The digest is calculated over the signature timestamp concatenated with the data portion of the message component. The inclusion of the signature timestamp in the digest assures against replay types of attacks. The exclusion of the filename and timestamp portions of the message component ensures that detached signatures are exactly the same as attached signatures prefixed to the message. Detached signatures are calculated on a separate file that has none of the message component header fields.

**Leading two octets of message digest:** To enable the recipient to determine if the correct public key was used to decrypt the message digest for authentication, by comparing this plaintext copy of the first two octets with the first two octets of the decrypted digest. These octets also serve as a 16-bit frame check sequence for the message.

**Key ID of sender's public key:** Identifies the public key that should be used to decrypt the message digest and, hence, identifies the private key that was used to encrypt the message digest.
Figure 1.3. General Format of PGP Message (from A to B)

The message component and optional signature component may be compressed using ZIP and may be encrypted using a session key.

The **session key component** includes the session key and the identifier of the recipient's public key that was used by the sender to encrypt the session key.
The entire block is usually encoded with radix-64 encoding.

**Key Rings:**
The key IDs are critical to the operation of PGP and two key IDs are included in any PGP message that provides both confidentiality and authentication. These keys need to be stored and organized in a systematic way for efficient and effective use by all parties. **The scheme used in PGP is to provide a pair of data structures at each node, one to store the public/private key pairs owned by that node and one to store the public keys of other users known at this node. These data structures are referred to, respectively, as the private-key ring and the public-key ring.**

Figure 1.4 shows the general structure of a private-key ring. We can view the ring as a table, in which each row represents one of the public/private key pairs owned by this user. Each row contains the following entries:

![Private-Key Ring](image)

![Public-Key Ring](image)

* = field used to index table

Figure 1.4. General Structure of Private- and Public-Key Rings
Timestamp: The date/time when this key pair was generated.

Key ID: The least significant 64 bits of the public key for this entry.

Public key: The public-key portion of the pair.

Private key: The private-key portion of the pair; this field is encrypted.

User ID: Typically, this will be the user's e-mail address (e.g. suresha@revainstitution.org). However, the user may choose to associate a different name with each pair or to reuse the same User ID more than once.

The private-key ring can be indexed by either User ID or Key ID.

Although it is intended that the private-key ring be stored only on the machine of the user that created and owns the key pairs, and that it be accessible only to that user, it makes sense to make the value of the private key as secure as possible. Accordingly, the private key itself is not stored in the key ring. Rather, this key is encrypted using CAST-128.

The procedure is as follows:

1. The user selects a passphrase to be used for encrypting private keys.

2. When the system generates a new public/private key pair using RSA, it asks the user for the passphrase. Using SHA-1, a 160-bit hash code is generated from the passphrase, and the passphrase is discarded.

3. The system encrypts the private key using CAST-128 with the 128 bits of the hash code as the key. The hash code is then discarded, and the encrypted private key is stored in the private-key ring.

Subsequently, when a user accesses the private-key ring to retrieve a private key, he or she must supply the passphrase. PGP will retrieve the encrypted private key, generate the hash code of the passphrase, and decrypt the encrypted private key using CAST-128 with the hash code.

This is a very compact and effective scheme. As in any system based on passwords, the security of this system depends on the security of the password. To avoid the temptation to write it down, the user should use a passphrase that is not easily guessed but that is easily remembered.

Figure 1.4 also shows the general structure of a public-key ring. This data structure is used to store public keys of other users that are known to this user.

Timestamp: The date/time when this entry was generated.
**Key ID:** The least significant 64 bits of the public key for this entry.

**Public Key:** The public key for this entry.

**User ID:** Identifies the owner of this key. Multiple user IDs may be associated with a single public key.

The public-key ring can be indexed by either User ID or Key ID

Now consider message transmission and reception using key rings. For simplicity, we ignore compression and radix-64 conversion in the following discussion.

First consider message transmission (Figure 1.5) and assume that the message is to be both signed and encrypted. The sending PGP entity performs the following steps

![PGP Message Generation](image)

**Figure 1.5. PGP Message Generation (from User A to User B, no compression or radix 64 conversion)**
1. Signing the message
   - PGP retrieves the sender's private key from the private-key ring using your_userid as an index. If your_userid was not provided in the command, the first private key on the ring is retrieved.
   - PGP prompts the user for the passphrase to recover the unencrypted private key.
   - The signature component of the message is constructed.

2. Encrypting the message
   - PGP generates a session key and encrypts the message.
   - PGP retrieves the recipient's public key from the public-key ring using her_userid as an index.
   - The session key component of the message is constructed.

The receiving PGP entity performs the following steps (Figure 1.6),

![Diagram of PGP Message Reception](image)

Figure 1.6. PGP Message Reception (from User A to User B, no compression or radix 64 conversion)
1. Decrypting the message
   - PGP retrieves the receiver's private key from the private-key ring, using the Key ID field in the session key component of the message as an index.
   - PGP prompts the user for the passphrase to recover the unencrypted private key.
   - PGP then recovers the session key and decrypts the message.

2. Authenticating the message
   - PGP retrieves the sender's public key from the public-key ring, using the Key ID field in the signature key component of the message as an index.
   - PGP recovers the transmitted message digest.
   - PGP computes the message digest for the received message and compares it to the transmitted message digest to authenticate.

Public-Key Management:
PGP has a clever, efficient, interlocking set of functions and formats to provide an effective confidentiality and authentication service. To complete the system, one final area needs to be addressed, that of public-key management. The PGP documentation captures the importance of this area:

   This whole business of protecting public keys from tampering is the single most difficult problem in practical public key applications. It is the "Achilles heel" of public key cryptography, and a lot of software complexity is tied up in solving this one problem.

PGP provides a structure for solving this problem, with several suggested options that may be used. Because PGP is intended for use in a variety of formal and informal environments with no rigid public-key management scheme is set up. The following methods are used for public key Management are The Use of Trust and Revoking Public Keys.

The Use of Trust:
PGP provide a convenient means of using trust, associating trust with public keys, and exploiting trust information.

Each entry in the public-key ring is a public-key certificate, associated with each entry is a **key legitimacy field** that indicates the extent to which PGP will trust that this is a valid public key for this user; the higher the level of trust, the stronger is the binding of this user ID
to this key. This field is computed by PGP. Also associated with the entry are zero or more signatures that the key ring owner has collected that sign this certificate. In turn, each signature has associated with it a **signature trust field** that indicates the degree to which this PGP user trusts the signer to certify public keys. The key legitimacy field is derived from the collection of signature trust fields in the entry. Finally, each entry defines a public key associated with a particular owner, and an **owner trust field** is included that indicates the degree to which this public key is trusted to sign other public-key certificates; this level of trust is assigned by the user.

The above three fields are each contained in a structure referred to as a trust flag byte. The content of this trust flag for each of these three uses is shown in Table 1.2

Figure 1.7 provides an example of the way in which signature trust and key legitimacy are related. It shows the structure of a public-key ring. The user has acquired a number of public keys, some directly from their owners and some from a third party such as a key server.

**Revoking Public Keys**

A user may wish to revoke his or her current public key either because compromise is suspected or simply to avoid the use of the same key for an extended period. Note that a compromise would require that an opponent somehow had obtained a copy of your unencrypted private key or that the opponent had obtained both the private key from your private-key ring and your passphrase.

The convention for revoking a public key is for the owner to issue a key revocation certificate, signed by the owner. This certificate has the same form as a normal signature certificate but includes an indicator that the purpose of this certificate is to revoke the use of this public key. Note that the corresponding private key must be used to sign a certificate that revokes a public key. The owner should then attempt to disseminate this certificate as widely and as quickly as possible to enable potential correspondents to update their public-key rings.
Figure 1.7. PGP Trust Model Example
S/MIME (Secure/ Multipurpose Internet Mail Extension):
S/MIME is a security enhancement to the MIME Internet e-mail format standard, based on technology from RSA Data Security. Both PGP and S/MIME are on an IETF standards track. S/MIME is the industry standard for commercial and organizational use, while PGP is the choice for personal e-mail security for many users. S/MIME is defined in a number of documents, most importantly RFCs 3369, 3370, 3850 and 3851.

To understand S/MIME, we need first to have a general understanding of the underlying e-mail format that it uses, namely MIME. But to understand the significance of MIME, we need to go back to the traditional e-mail format standard, RFC 822, which is still in common use. Accordingly, we discuss RFC 822, MIME and S/MIME.

RFC 822:

RFC 822 defines a format for text messages that are sent using electronic mail.

- It is the standard for Internet-based text mail message
- In the RFC 822 context, messages are viewed as having an envelope and contents.
- The envelope contains whatever information is needed to accomplish transmission and delivery.
- The contents compose the object to be delivered to the recipient.
- The RFC 822 standard applies only to the contents. However, the content standard includes a set of header fields that may be used by the mail system to create the envelope, and the standard is intended to facilitate the acquisition of such information by programs.

The overall structure of a message that conforms to RFC 822 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 header line usually consists of a keyword, followed by a colon, followed by the keyword's arguments; the format allows a long line to be broken up into several lines. The most frequently used keywords are From, To, Subject, and Date. Here is an example message:
Date: Tue, 16 Jan 1998 10:37:17 (EST)
From: "Suresha" <suresha@revainstitution.org>
Subject: The Syntax in RFC 822
To: edusatvtu@gmail.com
Cc: nalinaniranjan@hotmail.com

Hello. This section begins the actual message body, which is delimited from the message heading by a blank line.

Another field that is commonly found in RFC 822 headers is Message-ID. This field contains a unique identifier associated with this message.

**Multipurpose Internet Mail Extensions (MIME):**
MIME is an extension to the RFC 822 framework that is intended to address some of the problems and limitations of the use of SMTP (Simple Mail Transfer Protocol) or some other mail transfer protocol and RFC 822 for electronic mail.
The following are the limitations of the SMTP/822 scheme:

1. SMTP cannot transmit executable files or other binary objects. A number of schemes are in use for converting binary files into a text form that can be used by SMTP mail systems, including the popular UNIX UUencode/UUdecode scheme. However, none of these is a standard or even a de facto standard.

2. SMTP cannot transmit text data that includes national language characters because these are represented by 8-bit codes with values of 128 decimal or higher, and SMTP is limited to 7-bit ASCII.

3. SMTP servers may reject mail message over a certain size.

4. SMTP gateways that translate between ASCII and the character code EBCDIC do not use a consistent set of mappings, resulting in translation problems.
5. SMTP gateways to X.400 electronic mail networks cannot handle nontextual data included in X.400 messages.

6. Some SMTP implementations do not adhere completely to the SMTP standards defined in RFC 821. Common problems include:
   - Deletion, addition, or reordering of carriage return and linefeed
   - Truncating or wrapping lines longer than 76 characters
   - Removal of trailing white space (tab and space characters)
   - Padding of lines in a message to the same length
   - Conversion of tab characters into multiple space characters

MIME is intended to resolve these problems in a manner that is compatible with existing RFC 822 implementations. The specification is provided in RFCs 2045 through 2049.

The MIME specification includes the following elements
1. Five new message header fields are defined, which may be included in an RFC 822 header. These fields provide information about the body of the message.
2. A number of content formats are defined, thus standardizing representations that support multimedia electronic mail.
3. Transfer encodings are defined that enable the conversion of any content format into a form that is protected from alteration by the mail system.

In this subsection, we introduce the five message header fields. The next two subsections deal with content formats and transfer encodings.

The five header fields defined in MIME are as follows:

**MIME-Version:** Must have the parameter value 1.0. This field indicates that the message conforms to RFCs 2045 and 2046.

**Content-Type:** Describes the data contained in the body with sufficient detail that the receiving user agent can pick an appropriate agent or mechanism to represent the data to the user or otherwise deal with the data in an appropriate manner.
**Content-Transfer-Encoding:** Indicates the type of transformation that has been used to represent the body of the message in a way that is acceptable for mail transport.

**Content-ID:** Used to identify MIME entities uniquely in multiple contexts.

**Content-Description:** A text description of the object with the body; this is useful when the object is not readable (e.g., audio data).

**MIME Content Types:**

The bulk of the MIME specification is concerned with the definition of a variety of content types. This reflects the need to provide standardized ways of dealing with a wide variety of information representations in a multimedia environment.

Table 1.3 lists the content types specified in RFC 2046. There are seven different major types of content and a total of 15 subtypes. In general, a content type declares the general type of data, and the subtype specifies a particular format for that type of data.

<table>
<thead>
<tr>
<th>Type</th>
<th>Subtype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>Plain</td>
<td>Unformatted text; may be ASCII or ISO 8859.</td>
</tr>
<tr>
<td></td>
<td>Enriched</td>
<td>Provides greater format flexibility</td>
</tr>
<tr>
<td>Multipart</td>
<td>Mixed</td>
<td>The different parts are independent but are to be transmitted together.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>They should be presented to the receiver in the order that they appear in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the mail message.</td>
</tr>
<tr>
<td></td>
<td>Parallel</td>
<td>Differs from Mixed only in that no order is defined for delivering the parts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to the receiver.</td>
</tr>
<tr>
<td></td>
<td>Alternative</td>
<td>The different parts are alternative versions of the same information. They</td>
</tr>
<tr>
<td></td>
<td></td>
<td>are ordered in increasing faithfulness to the original, and the recipient's</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mail system should display the &quot;best&quot; version to the user.</td>
</tr>
<tr>
<td></td>
<td>Digest</td>
<td>Similar to Mixed, but the default type/subtype of each part is message/rfc822.</td>
</tr>
<tr>
<td>Message</td>
<td>rfc822</td>
<td>The body is itself an encapsulated message that conforms to RFC 822.</td>
</tr>
<tr>
<td></td>
<td>Partial</td>
<td>Used to allow fragmentation of large mail items, in a way that is transparent to the recipient.</td>
</tr>
<tr>
<td></td>
<td>External-body</td>
<td>Contains a pointer to an object that exists elsewhere.</td>
</tr>
</tbody>
</table>
Electronic Mail Security

VTU-EDUSAT Programme-16

Prof. Suresha, Dept of CSE, Reva ITM
2011-12

<table>
<thead>
<tr>
<th>Image</th>
<th>jpeg</th>
<th>The image is in JPEG format, JFIF encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>gif</td>
<td>The image is in GIF format.</td>
</tr>
<tr>
<td>Video</td>
<td>Mpeg</td>
<td>MPEG format.</td>
</tr>
<tr>
<td>Audio</td>
<td>Basic</td>
<td>Single-channel 8-bit ISDN mu-law encoding at a sample rate of 8 kHz.</td>
</tr>
<tr>
<td>Application</td>
<td>PostScript</td>
<td>Adobe Postscript.</td>
</tr>
<tr>
<td></td>
<td>octet-stream</td>
<td>General binary data consisting of 8-bit bytes.</td>
</tr>
</tbody>
</table>

MIME Transfer Encodings:

The other major component of the MIME specification is a definition of transfer encodings for message bodies. The objective is to provide reliable delivery across the largest range of environments.

The MIME standard defines two methods of encoding data. The Content-Transfer-Encoding field can actually take on six values, as listed in Table 1.4. However, three of these values (7bit, 8bit, and binary) indicate that no encoding has been done but provide some information about the nature of the data. For SMTP transfer, it is safe to use the 7bit form. The 8bit and binary forms may be usable in other mail transport contexts. Another Content-Transfer-Encoding value is x-token, which indicates that some other encoding scheme is used, for which a name is to be supplied. This could be a vendor-specific or application-specific scheme. The two actual encoding schemes defined are quoted-printable and base64. Two schemes are defined to provide a choice between a transfer technique that is essentially human readable and one that is safe for all types of data in a way that is reasonably compact.

**Table 1.4. MIME Transfer Encodings**

<table>
<thead>
<tr>
<th>7bit</th>
<th>The data are all represented by short lines of ASCII characters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>8bit</td>
<td>The lines are short, but there may be non-ASCII characters (octets with the high-order bit set).</td>
</tr>
<tr>
<td>binary</td>
<td>Not only may non-ASCII characters be present but the lines are not necessarily short enough for SMTP transport.</td>
</tr>
<tr>
<td>quoted-printable</td>
<td>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.</td>
</tr>
</tbody>
</table>
**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.

The **quoted-printable** transfer encoding is useful when the data consists largely of octets that correspond to printable ASCII characters. In essence, it represents nonsafe characters by the hexadecimal representation of their code and introduces reversible (soft) line breaks to limit message lines to 76 characters.

The **base64 transfer encoding**, also known as radix-64 encoding, is a common one for encoding arbitrary binary data in such a way as to be invulnerable to the processing by mail transport programs.

**Canonical Form:**

An important concept in MIME and S/MIME is that of canonical form. Canonical form is a format, appropriate to the content type that is standardized for use between systems. This is in contrast to native form, which is a format that may be peculiar to a particular system.

Table 1.5, from RFC 2049, should help clarify this matter.

**Table 1.5. Native and Canonical Form**

<table>
<thead>
<tr>
<th>Native Form</th>
<th>Canonical Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>The body to be transmitted is created in the system’s native format. The native character set is used and, where appropriate, local end-of-line conventions are used as well. The body may be a UNIX-style text file, or a Sun raster image, or a VMS indexed file, or audio data in a system-dependent format stored only in memory, or anything else that corresponds to the local model for the representation of some form of information. Fundamentally, the data is created in the &quot;native&quot; form that corresponds to the type specified by the media type.</td>
<td>The entire body, including &quot;out-of-band&quot; information such as record lengths and possibly file attribute information, is converted to a universal canonical form. The specific media type of the body as well as its associated attributes dictate the nature of the canonical form that is used.</td>
</tr>
</tbody>
</table>
Conversion to the proper canonical form may involve character set conversion, transformation of audio data, compression, or various other operations specific to the various media types. If character set conversion is involved, however, care must be taken to understand the semantics of the media type, which may have strong implications for any character set conversion.

**S/MIME Functionality:**

In terms of general functionality, S/MIME is very similar to PGP. Both offer the ability to sign and/or encrypt messages. In this subsection, we briefly summarize S/MIME capability. We then look in more detail at this capability by examining message formats and message preparation.

**Functions**

S/MIME provides the following functions:

- **Enveloped data:** This consists of encrypted content of any type and encrypted-content encryption keys for one or more recipients.

- **Signed data:** A digital signature is formed by taking the message digest of the content to be signed and then encrypting that with the private key of the signer. The content plus signature are then encoded using base64 encoding. A signed data message can only be viewed by a recipient with S/MIME capability.

- **Clear-signed data:** As with signed data, a digital signature of the content is formed. However, in this case, only the digital signature is encoded using base64. As a result, recipients without S/MIME capability can view the message content, although they cannot verify the signature.

- **Signed and enveloped data:** Signed-only and encrypted-only entities may be nested, so that encrypted data may be signed and signed data or clear-signed data may be encrypted.

**Cryptographic Algorithms:**

Table 1.6 summarizes the cryptographic algorithms used in S/MIME. S/MIME uses the following terminology, taken from RFC 2119 to specify the requirement level:
**Must:** The definition is an absolute requirement of the specification. An implementation must include this feature or function to be in conformance with the specification.

**Should:** There may exist valid reasons in particular circumstances to ignore this feature or function, but it is recommended that an implementation include the feature or function.

### Table 1.6. Cryptographic Algorithms Used in S/MIME

<table>
<thead>
<tr>
<th>Function</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUST support SHA-1. Receiver SHOULD support MD5 for backward compatibility. Sending and receiving agents MUST support DSS. Sending agents SHOULD support RSA encryption. Receiving agents SHOULD support verification of RSA signatures with key sizes 512 bits to 1024 bits.</td>
<td></td>
</tr>
<tr>
<td>Encrypt message digest to be used in forming a digital signature. Encrypt message digest to form digital signature.</td>
<td>MUST support SHA-1. Sending and receiving agents SHOULD support Diffie-Hellman. Sending and receiving agents MUST support RSA encryption with key sizes 512 bits to 1024 bits.</td>
</tr>
<tr>
<td>Encrypt session key for transmission with message.</td>
<td>Sending and receiving agents MUST support encryption with triple DES. Sending agents SHOULD support encryption with AES. Sending agents SHOULD support encryption with RC2/40.</td>
</tr>
<tr>
<td>Create a message authentication code</td>
<td>Receiving agents MUST support HMAC with SHA-1. Receiving agents SHOULD support HMAC with SHA-1.</td>
</tr>
</tbody>
</table>

**S/MIME incorporates three public-key algorithms:**

The Digital Signature Standard (DSS) is the preferred algorithm for digital signature. S/MIME lists Diffie-Hellman as the preferred algorithm for encrypting session keys. RSA, can be used for both signatures and session key encryption. For message encryption, three-key triple DES (tripleDES) is recommended.

The S/MIME specification includes a discussion of the procedure for deciding which content encryption algorithm to use. In essence, a sending agent has two decisions to make. First, the sending agent must determine if the receiving agent is capable of decrypting using a given encryption algorithm. Second, if the receiving agent is only capable of accepting weakly
encrypted content, the sending agent must decide if it is acceptable to send using weak encryption.

The following rules, in the following order, should be followed by a sending agent:
1. If the sending agent has a list of preferred decrypting capabilities from an intended recipient, it SHOULD choose the first (highest preference) capability on the list that it is capable of using.
2. If the sending agent has no such list of capabilities from an intended recipient but has received one or more messages from the recipient, then the outgoing message SHOULD use the same encryption algorithm as was used on the last signed and encrypted message received from that intended recipient.
3. If the sending agent has no knowledge about the decryption capabilities of the intended recipient and is willing to risk that the recipient may not be able to decrypt the message, then the sending agent SHOULD use tripleDES.
4. If the sending agent has no knowledge about the decryption capabilities of the intended recipient and is not willing to risk that the recipient may not be able to decrypt the message, then the sending agent MUST use RC2/40.

If a message is to be sent to multiple recipients and a common encryption algorithm cannot be selected for all, then the sending agent will need to send two messages. However, in that case, it is important to note that the security of the message is made vulnerable by the transmission of one copy with lower security.

S/MIME Messages:

S/MIME makes use of a number of new MIME content types, which are shown in Table 1.7. All of the new application types use the designation PKCS. This refers to a set of public-key cryptography specifications issued by RSA Laboratories and made available for the S/MIME effort.

<table>
<thead>
<tr>
<th>Type</th>
<th>Subtype</th>
<th>s/mime Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multipart</td>
<td>Signed</td>
<td></td>
<td>A clear-signed message in two parts: one is the message and the other is the signature</td>
</tr>
<tr>
<td>Application</td>
<td>pkcs 7-mime</td>
<td>signedData</td>
<td>A signed S/MIME entity.</td>
</tr>
<tr>
<td>Application</td>
<td>pkcs 7-mime</td>
<td>envelopedData</td>
<td>An encrypted S/MIME entity.</td>
</tr>
</tbody>
</table>
Securing a MIME Entity:

S/MIME secures a MIME entity with a signature, encryption, or both. A MIME entity may be an entire message (except for the RFC 822 headers), or if the MIME content type is multipart, then a MIME entity is one or more of the subparts of the message. The MIME entity is prepared according to the normal rules for MIME message preparation. Then the MIME entity plus some security-related data, such as algorithm identifiers and certificates, are processed by S/MIME to produce what is known as a PKCS object. A PKCS object is then treated as message content and wrapped in MIME.

Enveloped Data:

An application/pkcs7-mime subtype is used for one of four categories of S/MIME processing, each with a unique S/MIME-type parameter. In all cases, the resulting entity, referred to as an object, is represented in a form known as Basic Encoding Rules (BER), which is defined in ITU-T Recommendation X.209. The BER format consists of arbitrary octet strings and is therefore binary data. Such an object should be transfer encoded with base 64 in the outer MIME message. We first look at enveloped Data.

The steps for preparing an enveloped Data MIME entity are as follows:

1. Generate a pseudorandom session key for a particular symmetric encryption algorithm (RC2/40 or tripleDES).
2. For each recipient, encrypt the session key with the recipient's public RSA key.
3. For each recipient, prepare a block known as RecipientInfo that contains an identifier of the recipient's public-key certificate, an identifier of the algorithm used to encrypt the session key, and the encrypted session key.
4. Encrypt the message content with the session key.

The RecipientInfo blocks followed by the encrypted content constitute the envelopedData. This information is then encoded into base64.
A sample message (excluding the RFC 822 headers) is the following:

Content-Type: application/pkcs7-mime; smime-type=envelopeddata;
name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m
rfvbnj75.6tbBghyHhHUujhJhjH77n8HHGT9HG4VQpffyF467GhIGfHfYT6
7n8HHGghyHhHUujhJh4VQpffyF467GhIGfHfYT6jH7756tbB9H
f8HHGTfrvvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpffyF4
0GhIGfHfQbnj756YT64V

To recover the encrypted message, the recipient first strips off the base64 encoding. Then the recipient's private key is used to recover the session key. Finally, the message content is decrypted with the session key.

**SignedData:**

The signedData smime-type can actually be used with one or more signers. For clarity, we confine our description to the case of a single digital signature. The steps for preparing a signedData MIME entity are as follows:

1. Select a message digest algorithm (SHA or MD5).
2. Compute the message digest, or hash function, of the content to be signed.
3. Encrypt the message digest with the signer's private key.
4. Prepare a block known as SignerInfo that contains the signer's public-key certificate, an identifier of the message digest algorithm, an identifier of the algorithm used to encrypt the message digest, and the encrypted message digest.

The signedData entity consists of a series of blocks, including a message digest algorithm identifier, the message being signed, and SignerInfo. The signedData entity may also include a set of public-key certificates sufficient to constitute a chain from a recognized root or top-level certification authority to the signer. This information is then encoded into base64. A sample message (excluding the RFC 822 headers) is the following:

Content-Type: application/pkcs7-mime; smime-type=signed-data;
name=smime.p7m
Content-Transfer-Encoding: base64
To recover the signed message and verify the signature, the recipient first strips off the base64 encoding. Then the signer's public key is used to decrypt the message digest. The recipient independently computes the message digest and compares it to the decrypted message digest to verify the signature.

**ClearSigning:**
Clear signing is achieved using the multipart content type with a signed subtype. This signing process does not involve transforming the message to be signed, so that the message is sent "in the clear." Thus, recipients with MIME capability but not S/MIME capability are able to read the incoming message.

A multipart/signed message has two parts. The first part can be any MIME type but must be prepared so that it will not be altered during transfer from source to destination. This means that if the first part is not 7bit, then it needs to be encoded using base64 or quoted-printable. Then this part is processed in the same manner as signedData, but in this case an object with signedData format is created that has an empty message content field. This object is a detached signature. It is then transfer encoded using base64 to become the second part of the multipart/signed message. This second part has a MIME content type of application and a subtype of pkcs7-signature. Here is a sample message:

```
Content-Type: multipart/signed;
protocol="application/pkcs7-signature";
micalg=sha1; boundary=boundary42
boundary42
Content-Type: text/plain
This is a clear-signed message.
boundary42
```
The protocol parameter indicates that this is a two-part clear-signed entity. The micalg parameter indicates the type of message digest used. The receiver can verify the signature by taking the message digest of the first part and comparing this to the message digest recovered from the signature in the second part.

**Registration Request:**

Typically, an application or user will apply to a certification authority for a public-key certificate. The application/pkcs10 S/MIME entity is used to transfer a certification request. The certification request includes certificationRequestInfo block, followed by an identifier of the public-key encryption algorithm, followed by the signature of the certificationRequestInfo block, made using the sender's private key. The certificationRequestInfo block includes a name of the certificate subject (the entity whose public key is to be certified) and a bit-string representation of the user's public key.

**Certificates-Only Message:**

A message containing only certificates or a certificate revocation list (CRL) can be sent in response to a registration request. The message is an application/pkcs7-mime type/subtype with an smime-type parameter of degenerate. The steps involved are the same as those for creating a signedData message, except that there is no message content and the signerInfo field is empty.
S/MIME Certificate Processing:
S/MIME uses public-key certificates that conform to version 3 of X.509. The key-management scheme used by S/MIME is in some ways a hybrid between a strict X.509 certification hierarchy and PGP's web of trust. As with the PGP model, S/MIME managers and/or users must configure each client with a list of trusted keys and with certificate revocation lists. That is, the responsibility is local for maintaining the certificates needed to verify incoming signatures and to encrypt outgoing messages. On the other hand, the certificates are signed by certification authorities.

User Agent Role:
An S/MIME user has several key-management functions to perform:

Key generation: The user of some related administrative utility (e.g., one associated with LAN management) MUST be capable of generating separate Diffie-Hellman and DSS key pairs and SHOULD be capable of generating RSA key pairs. Each key pair MUST be generated from a good source of nondeterministic random input and be protected in a secure fashion. A user agent SHOULD generate RSA key pairs with a length in the range of 768 to 1024 bits and MUST NOT generate a length of less than 512 bits.

Registration: A user's public key must be registered with a certification authority in order to receive an X.509 public-key certificate.

Certificate storage and retrieval: A user requires access to a local list of certificates in order to verify incoming signatures and to encrypt outgoing messages. Such a list could be maintained by the user or by some local administrative entity on behalf of a number of users.

VeriSign Certificates:
There are several companies that provide certification authority (CA) services. For example, Nortel has designed an enterprise CA solution and can provide S/MIME support within an organization. There are a number of Internet-based CAs, including VeriSign, GTE, and the U.S. Postal Service. Of these, the most widely used is the VeriSign CA service, a brief description of which we now provide. VeriSign provides a CA service that is intended to be compatible with S/MIME and a variety of other applications. VeriSign issues X.509 certificates with the product name VeriSign Digital ID. As of early 1998, over 35,000
commercial Web sites were using VeriSign Server Digital IDs, and over a million consumer Digital IDs had been issued to users of Netscape and Microsoft browsers.

The information contained in a Digital ID depends on the type of Digital ID and its use. At a minimum, each Digital ID contains

- Owner's public key
- Owner's name or alias
- Expiration date of the Digital ID
- Serial number of the Digital ID
- Name of the certification authority that issued the Digital ID
- Digital signature of the certification authority that issued the Digital ID

Digital IDs can also contain other user-supplied information, including

- Address
- E-mail address
- Basic registration information (country, zip code, age, and gender)

VeriSign provides three levels, or classes, of security for public-key certificates, as summarized in Table 1.8. A user requests a certificate online at VeriSign's Web site or other participating Web sites. Class 1 and Class 2 requests are processed on line, and in most cases take only a few seconds to approve. Briefly, the following procedures are used:

- For Class 1 Digital IDs, VeriSign confirms the user's e-mail address by sending a PIN and Digital ID pick-up information to the e-mail address provided in the application.
- For Class 2 Digital IDs, VeriSign verifies the information in the application through an automated comparison with a consumer database in addition to performing all of the checking associated with a Class 1 Digital ID. Finally, confirmation is sent to the specified postal address alerting the user that a Digital ID has been issued in his or her name.
- For Class 3 Digital IDs, VeriSign requires a higher level of identity assurance. An individual must prove his or her identity by providing notarized credentials or applying in person.

Table 1.8. VeriSign Public-Key Certificate Classes
<table>
<thead>
<tr>
<th>Class</th>
<th>Summary of Confirmation of Identity</th>
<th>IA Private Key Protection</th>
<th>Certificate Applicant and Subscriber Private Key Protection</th>
<th>Applications implemented or contemplated by Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 1</td>
<td>Automated unambiguous name and e-mail address search</td>
<td>PCA: trustworthy hardware; CA: trustworthy software or trustworthy hardware</td>
<td>Encryption software (PIN protected) recommended but not required</td>
<td>Web-browsing and certain e-mail usage</td>
</tr>
<tr>
<td>Class 2</td>
<td>Same as Class 1, plus automated enrollment information check plus automated address check</td>
<td>PCA and CA: trustworthy hardware</td>
<td>Encryption software (PIN protected) required</td>
<td>Individual and intra and inter-company E-mail, online subscriptions, password replacement, and software validation</td>
</tr>
<tr>
<td>Class 3</td>
<td>Same as Class 1, plus personal presence and ID documents plus Class 2 automated ID check for individuals; business records (or filings) for organizations</td>
<td>PCA and CA: trustworthy hardware</td>
<td>Encryption software (PIN protected) required; hardware token recommended but not required</td>
<td>E-banking, corp, database access, personal banking, membership-based online services, content integrity services, e-commerce server, software validation; authentication of LRAAs; and strong encryption for certain servers</td>
</tr>
</tbody>
</table>

**Enhanced Security Services:**

Three enhanced security services have been proposed in an Internet draft. The details of these may change, and additional services may be added. The three services are as follows:

- **Signed receipts:** A signed receipt may be requested in a SignedData object. Returning a signed receipt provides proof of delivery to the originator of a message and allows the originator to demonstrate to a third party that the recipient received the message. In essence, the recipient signs the entire original message plus original (sender's) signature and appends the new signature to form a new S/MIME message.
• **Security labels:** A security label may be included in the authenticated attributes of a SignedData object. A security label is a set of security information regarding the sensitivity of the content that is protected by S/MIME encapsulation. The labels may be used for access control, by indicating which users are permitted access to an object. Other uses include priority (secret, confidential, restricted, and so on) or role based, describing which kind of people can see the information (e.g., patient's healthcare team, medical billing agents, etc.).

• **Secure mailing lists:** When a user sends a message to multiple recipients, a certain amount of per-recipient processing is required, including the use of each recipient's public key. The user can be relieved of this work by employing the services of an S/MIME Mail List Agent (MLA). An MLA can take a single incoming message, perform the recipient-specific encryption for each recipient, and forward the message. The originator of a message need only send the message to the MLA, with encryption performed using the MLA's public key.

References:


For any Clarifications, Send queries to
suresha@revainstitution.org
suresha_rec@rediffmail.com