

## PGP and S/MIME. An Overview

### PGP Services

#### - Authentication:

1. The sender creates a msg
2. SHA-1 is used to generate 160 bit hash
3. SHA-1 generated hash is encrypted with RSA using the sender's private key, and prepended to msg
4. The receiver uses RSA with the sender's public key to decrypt and recover the hash code
5. The receiver generates the SHA-1 from the message received, and compares it with the decrypted one

Signatures can either be part of the message or be detached from the message, as an attachment.

#### - Confidentiality:

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

Other methods that can be used with PGP are Diffie-Hellman (only for sig) or an alternative to DH known as El-Gamal (both sig and encryption).

#### - Confidentiality and Authentication:

1. Create a sig for the plaintext msg and prepend to the message
2. Plaintext msg plus sig is encrypted, using CAST-128 (or IDEA, TDEA) and the session key is encrypted with RSA or El-Gamal

In general when both services are used : The sender signs the msg, encrypts both msg and plaintext with session key and finally encrypts the session key with the recipient's public key.

#### - Compression:

Due to the nature of the compression algorithm used in PGP (non-deterministic) PGP first signs the document and then compresses it. Encryption is done after compression, thus making the msg containing less redundant information. The compression issue is also because of compatibility. This way different versions of PGP can use different versions of compression algos but there is always compatibility.

#### - E-mail compatibility:

When the plaintext is encrypted, the e-mail body consists of a stream of arbitrary 8-bit octets.

Because many e-mail systems only permit the use of blocks consisting of ASCII text, PGP encodes the resulting msg from raw 8-bit to ASCII text. To accommodate this is using radix-64 conversion.

Thus the receiving end will convert the msg from base-64 to raw 8-bit before performing further processing.

## S/MIME Overview

A few words on MIME:

- MIME extends the capabilities of RFC822 to carry non-textual content, non-ASCII character sets, long messages
- Uses extra header fields in RFC 822 e-mails to specify form and content of extensions
- Supports a variety of content types (still ASCII coded for compatibility)
- MIME specifies 5 new headers:
  - MIME-version
  - Content-type (ps, app, audio, image, video etc.).  
Multipart/mixed is the most important. Each part can be a separate MIME message (nesting is possible). Parts separated by a boundary string defined in content-type
  - Content-transfer-encoding. Indicates how data was encoded from raw to ASCII (base64 or radix64 is a common one)
  - Content-ID (optional)
  - Content-Description (optional)
- S/MIME
  - S/MIME adds security features to MIME (hence the S)
  - S/MIME adds 5 new content-type/subtype combinations like:
    - application/pkcs7-mime; smime-type-enveloped data
    - application/pkcs7-mime; smime-type-signed-data
    - multipart/signed
  - S/MIME processing can be applied to any MIME entity (even on a part of multipart that is already S/MIME <- smime-content-type). Hence encryption and sig can be applied one after another and in either order.
- S/MIME processing

MIME\_entity -- (S/MIME processing) --> PKCS\_object --- (Base64 encoding) -->S/MIMEentity

- Initial S/MIME produces a PKCS object (PKCS- Public Key Cryptogr. Std)
- PKCS object include information needed for processing by recipient as well as the original content
- PKCS objects are in binary format, hence need base64 encoding to produce final result MIME object of smime-content-type
- Recipient reverses steps (Final MIME entity may be of S/MIME-content-type; further S/MIME processing)
- S/MIME-content-types (some of them)
  - type = Multipart; subtype = Signed (a clear-signed msg and the msg)
  - type = Application; subtype = pkcs7-mime           signedData
  - type = Application; subtype = pkcs7-mime           envelopdData
  - type = Application; subtype = pkcs7-mime           degenerate signedData

#### S/MIME envelopedData (Confidentiality & authentication):

- Generate a session key for symmetric encryption (RC2/40, 3DES)
- Encrypt the session key with the recipient's public RSA key
- Prepare RecipientInfo block: Contains the sender's Certificate, Algorithm Ids, and the encrypted session key
- Encrypt the message content (MIME object) with the session key (EncryptedContent)
- Prepare EncryptedContentInfo block: Symmetric algorithm Ids, EncryptedContent..
- *Prepare EnvelopdData* PKCS object: RecipientInfo || EncryptedContentInfo
- Base64 encode the PKCS object and attach S/MIME header  
*content-type: application/pkcs7-mime; smime-type=enveloped-data*
- Recipient reverses steps. Gets algo Info from RecipientInfo block

#### S/MIME signedData (authenticity, non-repudiation)

- Used to sign MIME entities
- Hash MIME entity and sign hash value using private key
- Prepare *SignerInfo* block: Signer's cert, hash and sig algorithm ID, hash value and signature
- *Prepare SignedData* PKCS object: SignerInfo || original MIME entity
- Base64 encode the PKCS object and attach S/MIME header  
*content-type: application/pkcs7-mime; smime-type=signed-data*

*\*Multiple signers supported*

#### S/MIME clear signing

- Used to clear sign message (multipart in two parts: msg and sig)
- First Part contains MIME entity to sign
- Second part contains S/MIME  
*content-type: application/pkcs7-signature (created as signed-data)*
- Recipients who have MIME but not S/MIME can still read the msg
- Recipients who have S/MIME use the first part as MIME object in S/MIME signature verification

#### Algorithms supported (S/MIME):

- Symmetric: DES, 3DES, RC2/40 and 64 bit keys
- Public key encryption: RSA, El-Gamal
- Hashing: SHA-1, MD5
- Signature: RSA, DSS

#### Algorithms supported (PGP):

- Symmetric: DES, 3DES, RC2/40 and 64 bit keys, AES and others
- Public key encryption: RSA, El-Gamal
- Hashing: SHA-1, MD5 and others
- Signature: RSA, DSS, ECDSA and others

## Key Management (PGP & S/MIME)

- Both PGP and S/MIME must store a pair of public/private key
- Common problem is where are these keys kept (user workstation)
- And where these keys come from
- PGP uses passphrase to decrypt the private key

### S/MIME key management

- Uses public key certificates and certificate chains to validate keys
- X.509 certificate specifies the use of the key
- Roots of Trust are embedded in MUA
- Revocation. How to communicate this?
- Certificate registration assurance

### PGP key management

- Completely different trust model (web of trust)
- No centralised authority like root CA
- Individuals sign one another's keys. This is stored along with keys in key ring
- PGP computes a trust level for each public key in key-ring
- Users interpret trust levels for themselves
- Trust levels depend on number of signatures on the key and trust level accorded to each of these signatures
- It is difficult to set-up a large web-of-trust
- Later versions of PGP support X.509 certificates