Solutions to Selected Problems

Guide to Internet Cryptography

Companion Material

February 18, 2026

Preface This document provides solutions to selected problems from the book *Guide to Internet Cryptography: Security Protocols and Real-World Attack Implications*. The material is intended for educational use in courses and self-study.

Book website: https://link.springer.com/book/10.1007/978-3-031-19439-9

1 Chapter 17: S/MIME

Problem 17.1 SMTP and RFC 822

(a) Can you use German umlauts like ä, ö, ü in e-mail addresses? If yes, where?

(b) Which domain must be queried for the DNS MX record? Where can you find this domain in the e-mail source code?

(c) Look at the source code of one of your e-mails. Can you separate the static RFC 822 headers from the ones generated during SMTP transmission?

Solution

(a) Yes. RFC 6530 defines a framework how to fully internationalize email addresses, and points to the different RFCs containing the details.

(b) The domains in the domain parts of all email addresses defining the recipients of the email must be queried for the `MX` RR. These email addresses can be found in the `TO`, the `CC` and the `BCC` headers, and maybe some other headers.

(c) This is a practical task.

Problem 17.2 PEM

(a) MIME and PEM use RFC 822 headers to encode information about the content. Can you describe the differences between these two concepts?

(b) Why does base64 encoding increase content size by one-third?

Solution

(a) MIME uses RFC 822 headers to encode metadata about the content, like MIME type and transport encoding. PEM uses such headers to encode cryptographic metadata. For

MIME, cryptographic metadata is einter contained in OpenPGP or CMS data structures.

(b) Because each 3 bytes are encoded as 4 bytes.

Problem 17.3 MIME

(a) Which are the two most crucial MIME header fields? How can you distinguish MIME headers from other RFC 822 headers?

(b) What is the MIME type of a Microsoft Word file?

(c) Encode the name of the danish philosopher *Søren Kirgegaard* in quoted-printable.

(d) Encode the ASCII sequence `abcd` in base 64.

(e) Choose one of your e-mails and analyze the source code. Can you sketch the MIME tree of this e-mail?

(f) If you have an HTML text with German umlauts, do you need quoted-printable encoding for this?

Solution

(a) The two most crucial MIME headers are `Content-Type` and `Transfer-Encoding`. The guarantee safe SMTP transmission, and give hints to the recipient's client on how to handle data. MIME only defines 5 headers, so they are easy to recognize. In addition, MIME headers may also occur in the RFC 822 body of an email.

(b) The MIME type of a Microsoft Word file depends on the file format:
For modern Word Documents (.docx):
`application/vnd.openxmlformats-officedocument.wordprocessingml.document`
This is for Word documents created in Word 2007 and later, which use the Office Open XML format.
For legacy Word Documents (.doc):
`application/msword`
This is for older Word documents (Word 97-2003) using the binary .doc format.

(c) Step 1: Convert ASCII to Binary Each ASCII character is 8 bits:

$$\begin{aligned}
\mathtt{a} &= \text{0x61} = \mathtt{01100001}\\
\mathtt{b} &= \text{0x62} = \mathtt{01100010}\\
\mathtt{c} &= \text{0x63} = \mathtt{01100011}\\
\mathtt{d} &= \text{0x64} = \mathtt{01100100}
\end{aligned}$$

Combined 32-bit sequence:

$$\mathtt{01100001\ 01100010\ 01100011\ 01100100} \tag{1}$$

Step 2: Group into 6-bit Chunks (Sextets) Base64 encodes data in 6-bit groups. We divide the 32-bit sequence:

$$\underbrace{\texttt{011000}}_{\text{sextet 1}}\ \underbrace{\texttt{010110}}_{\text{sextet 2}}\ \underbrace{\texttt{001001}}_{\text{sextet 3}}\ \underbrace{\texttt{100011}}_{\text{sextet 4}}\ \underbrace{\texttt{011001}}_{\text{sextet 5}}\ \underbrace{\texttt{00}}_{\text{2 bits}} \tag{2}$$

Since we have 32 bits, we get $\lceil 32/6 \rceil = 6$ sextets. The last sextet only contains 2 bits from our data, so we **pad with 4 zeros** on the right:

$$\texttt{011000} \mid \texttt{010110} \mid \texttt{001001} \mid \texttt{100011} \mid \texttt{011001} \mid \texttt{00}\underline{\texttt{0000}} \tag{3}$$

Step 3: Convert Sextets to Base64 Characters Convert each 6-bit value to decimal and look up in the Base64 alphabet (A–Z: 0–25, a–z: 26–51, 0–9: 52–61, +: 62, /: 63):

Binary (6 bits)	Decimal	Base64 Character
`011000`	24	`Y`
`010110`	22	`W`
`001001`	9	`J`
`100011`	35	`j`
`011001`	25	`Z`
`000000`	0	`A`

Step 4: Add Padding Base64 output length must be a multiple of 4 characters. Since we have 6 sextets but only $32/6 = 5.\overline{3}$ sextets of actual data, the last sextet contains padding bits. We need to add **padding characters** (`=`) to reach a multiple of 4.
With 4 input bytes (32 bits), the Base64 output should be:

$$\left\lceil \frac{4 \text{ bytes} \times 8}{6} \right\rceil = \left\lceil \frac{32}{6} \right\rceil = 6 \text{ characters} \tag{4}$$

Rounded up to the nearest multiple of 4: $\lceil 6/4 \rceil \times 4 = 8$ characters.
So we have 6 encoded characters and need $8 - 6 = 2$ padding characters:

$$\boxed{\texttt{YWJjZA==}} \tag{5}$$

Verification We can verify this is correct:

- Input: 4 bytes = 32 bits
- Output: 8 Base64 characters (6 data + 2 padding)
- Formula: $\lceil n/3 \rceil \times 4$ where $n = 4$ bytes $\Rightarrow \lceil 4/3 \rceil \times 4 = 2 \times 4 = 8$

(e) This is a practical excercise.

(f) No, you can also use HTML entities.

Problem 17.4 PKCS#7 and CMS

(a) What is the main difference between ASN.1 in BER encoding on the one side and XML and JSON on the other?

(b) Can you find other data formats besides X.509 and PKCS#7/CMS that use ASN.1 as the specification language?

(c) Why does `SignerInfo` contain a complete certificate, but `RecipientInfo` only a pointer to a certificate?

(d) Why can there be more than one hash algorithm in `DigestAlgorithmIdentifiers`?

Solution

(a) BER encoding uses the Tag/Length/Value paradigm, there is no need for special characters. XML and JSON both use special characters (<, >, {, }, ...) the delimit parts of complex data structures.

(b) Telecommunications: 3GPP/LTE/5G signaling protocols
LDAP: Directory services
SNMP: Network management

(c) To verify a signature, the complete certificate is needed: The public key from the certificate is needed to verify the signature, and the signature (and all other components) is needed to verify the certificate chain up to the root certificate.
To decrypt a ciphertext, the recipient only needs to know which private key (locally stored) must be used, therefore a pointer to the certificate is a pointer to this private key.

(d) Because in PKCS#7/CMS, more than one signature may be attached to a message. The two or more signatures may use different hash algorithms. In an email context, this is also possible, but a valid usecase is hard to imagine.

Problem 17.5 S/MIME

(a) Which MIME content types do the two leaves of a `multipart/signed` entity have if the content was first encrypted and then signed?

(b) If an encrypted MIME entity is also signed, does this mean that the ciphertext has not been altered after encryption?

(c) Can you forward signed or encrypted e-mails? Does this make sense?

(d) Suppose you, Mallory, intercept an e-mail that was encrypted and then signed by Alice and addressed to Bob. (How) Can you make Bob believe that this e-mail originated from you?

Solution

(a)

```
Content-Type: multipart/signed;
    protocol="application/pkcs7-signature";
    micalg=sha-256;
    boundary="boundary"

--boundary
Content-Type: application/pkcs7-mime;
    smime-type=enveloped-data;
    name="smime.p7m"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7m"

[Base64-encoded PKCS#7 EnvelopedData]
--boundary
Content-Type: application/pkcs7-signature;
    name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"

[Base64-encoded PKCS#7 signature]
--boundary--
--boundary--
```

(b) No, because the ciphertext could have been altered and then re-signed.

(c) You can forward a signed email; here, the signature may not cover the whole MIME body and the email client may run into problems deciding what to display as a result of signature verification. You cannot simply forward an encrypted email - the recipient will most probably not be able to decrypt it. The only exception may be if the recipient was one of the original recipients of the forwarded email.

(d) Simply remove the signature, and sign the ciphertext yourself.

Problem 17.6 S/MIME encryption

(a) Why do S/MIME e-mails addressed to two recipients contain three `RecipientInfo` CMS objects?

(b) What is the ASN.1 object identifier (OID) for AES?

(c) Is there any real-world use case where only a part of the MIME tree is encrypted?

(d) Why are always complete MIME entities encrypted? Do the resulting ciphertexts contain known plaintext?

Solution

(a) Because the sender also must be able to decrypt the email, which is typically stored in some `Sent` mailbox.

(b) AES does not have a single OID — instead, there are **different OIDs for each key size and mode of operation**.

AES with CBC Mode These are the most commonly used in cryptographic standards like PKCS and S/MIME:

Algorithm	OID	Arc Notation
AES-128-CBC	2.16.840.1.101.3.4.1.2	`aes128-CBC`
AES-192-CBC	2.16.840.1.101.3.4.1.22	`aes192-CBC`
AES-256-CBC	2.16.840.1.101.3.4.1.42	`aes256-CBC`

AES with ECB Mode

Algorithm	OID	Arc Notation
AES-128-ECB	2.16.840.1.101.3.4.1.1	`aes128-ECB`
AES-192-ECB	2.16.840.1.101.3.4.1.21	`aes192-ECB`
AES-256-ECB	2.16.840.1.101.3.4.1.41	`aes256-ECB`

AES with Other Modes Other modes (OFB, CFB, GCM, CCM, etc.) also have their own OIDs following similar patterns:

Algorithm	OID
AES-128-GCM	2.16.840.1.101.3.4.1.6
AES-192-GCM	2.16.840.1.101.3.4.1.26
AES-256-GCM	2.16.840.1.101.3.4.1.46

Complete Reference Table For completeness, here is a table showing the pattern across all three key sizes:

Mode	AES-128	AES-192	AES-256
ECB	2.16.840.1.101.3.4.1.1	2.16.840.1.101.3.4.1.21	2.16.840.1.101.3.4.1.41
CBC	2.16.840.1.101.3.4.1.2	2.16.840.1.101.3.4.1.22	2.16.840.1.101.3.4.1.42
OFB	2.16.840.1.101.3.4.1.3	2.16.840.1.101.3.4.1.23	2.16.840.1.101.3.4.1.43
CFB	2.16.840.1.101.3.4.1.4	2.16.840.1.101.3.4.1.24	2.16.840.1.101.3.4.1.44
GCM	2.16.840.1.101.3.4.1.6	2.16.840.1.101.3.4.1.26	2.16.840.1.101.3.4.1.46
CCM	2.16.840.1.101.3.4.1.7	2.16.840.1.101.3.4.1.27	2.16.840.1.101.3.4.1.47
Wrap	2.16.840.1.101.3.4.1.5	2.16.840.1.101.3.4.1.25	2.16.840.1.101.3.4.1.45

(c) Not really, because email alwas need to be re-encrypted before being forwarded or replied to. In these cases, it only makes sense to encrypte the wole body.

(d) Encrypting and decrypting complete MIME entities is consistent with the philosophy of MIME to attack metadata to each MIME entity. This also preservers the structure of a MIME tree. Each MIME entity has known header types, which may serve as known plaintext in attacks.

Problem 17.7 S/MIME signature

(a) What would happen if you open an e-mail with two `SignerInfo` CMS objects?

(b) What is the ASN.1 object identifier (OID) for the RSA-PKCS#1 v 1.5 signature algorithm?

(c) Is there any real-world use case where only a part of the MIME tree is signed?

(d) Why must the leaves of a MIME tree be canonicalized before computing the digital signature?

(e) Suppose you use an outdated MIME e-mail client which is not S/MIME compliant. Which of the two signature formats would allow you to have the message displayed?

Solution

(a) The email client would be confused on how to handle this situation, since most clients only have a Boolean display for valid/invalid signatures.

(b) The ASN.1 OID for RSA-PKCS#1 v1.5 signature algorithms depends on the **hash function** used with RSA.

RSA-PKCS#1 v1.5 Signature OIDs Most Common (with SHA-2 family)

Algorithm	OID	Arc Notation
sha256WithRSAEncryption	1.2.840.113549.1.1.11	`pkcs-1.11`
sha384WithRSAEncryption	1.2.840.113549.1.1.12	`pkcs-1.12`
sha512WithRSAEncryption	1.2.840.113549.1.1.13	`pkcs-1.13`

Legacy (with SHA-1 and MD5)

Algorithm	OID	Arc Notation
sha1WithRSAEncryption	1.2.840.113549.1.1.5	`pkcs-1.5`
md5WithRSAEncryption	1.2.840.113549.1.1.4	`pkcs-1.4`

Additional SHA-2 Variants

Algorithm	OID	Arc Notation
sha224WithRSAEncryption	1.2.840.113549.1.1.14	`pkcs-1.14`
sha512-224WithRSAEncryption	1.2.840.113549.1.1.15	`pkcs-1.15`
sha512-256WithRSAEncryption	1.2.840.113549.1.1.16	`pkcs-1.16`

With SHA-3

Algorithm	OID
id-rsassa-pkcs1-v1_5-with-sha3-224	2.16.840.1.101.3.4.3.13
id-rsassa-pkcs1-v1_5-with-sha3-256	2.16.840.1.101.3.4.3.14
id-rsassa-pkcs1-v1_5-with-sha3-384	2.16.840.1.101.3.4.3.15
id-rsassa-pkcs1-v1_5-with-sha3-512	2.16.840.1.101.3.4.3.16

Plain RSA Encryption OID Note that there is also an OID for **RSA encryption** (not signature):

- **rsaEncryption**: `1.2.840.113549.1.1.1`

This OID is used in public key fields (like in X.509 certificates) to indicate the key type, but **not** in signature algorithm fields.

Most Common in Modern Usage In current X.509 certificates and CMS/PKCS#7 signatures, you will most commonly see:

- `1.2.840.113549.1.1.11` (SHA-256 with RSA)
- `1.2.840.113549.1.1.12` (SHA-384 with RSA)

- `1.2.840.113549.1.1.13` (SHA-512 with RSA)

The SHA-1 variant (`1.2.840.113549.1.1.5`) is deprecated and should no longer be used for new signatures.

(c) Yes, forwarding of signed emails.

(d) Without canonicalization, any change to the byte sequence occurring during transport may result in an invalid signature.

(e) The `multipart/signed` format.

Problem 17.8 PGP/MIME

Is the first part of `multipart/encrypted` a ciphertext?

Solution

No, it is just a placeholder to make the PGP/MIME structure for encryption similar to the one used for signatures.