DANE SMIMEA Extension – Developer's Implementation Guide

Toolkits

The S/MIME MUA extension depends on several toolkits. Depending on the environment, some of the toolkits may be implemented within the MUA, while others call out to external libraries. The programming for S/MIME is quite complex, but fortunately most of the complexity can be handled by existing libraries.

DNS queries

MUAs typically provide no support for DNS lookups beyond those needed to resolve names in e-mail addresses and URLs. Hence the extension needs an external DNS library, both to query for SMIMEA records, and to validate DNSSEC signatures. While it would be possible in theory to do the DNSSEC validation within the extension code, the validation process is quite complex and hard to debug, so we don't recommend it. A good candidate is getdns, described at http://getdnsapi.net/. It's written in C but comes with a python interface that may be useful for prototyping or debugging.

Cryptography

S/MIME requires an extensive set of cryptographic operations for key generation and checking, encryption and decryption, creating and checking signatures, and creating and parsing the ASN.1 data structures used in the Cryptographic Message Syntax (CMS). Fortunately, libraries are available that handle all of the difficult parts and present a high level interface to applications. By far the most popular is Openssl at www.openssl.org (Apache license). It's also written in C. There is a python interface library but it's incomplete and doesn't handle the CMS or PKCS#7 functions needed for S/MIME. The descriptions below use openssl functions and command-line applications as examples.

MIME and message headers

All S/MIME messages are structured using MIME (Multipurpose Internet Mail Extensions) encoding. The extension needs to be able to create MIME messages from chunks of signed or encrypted data, and to parse and extract MIME parts in messages, including retrieving the MIME headers for each part. The MUA will generally have MIME facilities since they're needed to handle attachments and formatted mail, so the extension should be able to use the existing facilities. In some cases, openssl handles MIME creation and parsing for the parts specific to signed or encrypted messages.

The extension also needs to be able to retrieve the sender and recipient addresses from the message's To: and From: headers. In most cases the MUA will provide those facilities, but if it doesn't, picking the

address out of the header is straightforward:

1. Delete any strings surrounded by parentheses (). Nested parentheses are allowed, so the code has to count the parenthesis nesting level to track what is a comment, e.g. (able (baker)) is one comment. Also delete any white space adjacent to the deleted comments.
2. If there is a string surrounded by angle brackets < >, that string without the angle brackets is the address and any other text is comments. Otherwise, whatever remains after deleting the comments is the address.
3. If a To: header has multiple recipients, the addresses are separated by commas. After deleting the parenthesized comments, treat the string between commas as an address. For example, in this header:

To: (robert) bob@bob.com, frederick <fred@fred.com>

First delete the parenthesized comment "(robert) ". Since there are no angle brackets before the first comma, the first address is bob@bob.com. The second address is fred@fred.com, with the text "frederick" outside the angle brackets also being a comment.

Create a signing or encryption certificate

Certificates are generally created by using the OpenSSL command line utility.

Create a certificate

1. Create a new RSA private key of 2048 bits:

```
$ openssl genrsa -out certkey.pem 2048
```

2. Create the private key and corresponding public key in DER (binary) form:

```
$ openssl rsa -in certkey.pem -outform der -out certkey.der
$ openssl rsa -in certkey.pem -pubout -outform der -out certpub.der
```

3. Create a self-signed certificate using that key. The user name here is "Sample
4. Person" and her e-mail address is "sample@example.org":

```
$ openssl req -subj "/CN=Sample Person/emailAddress=sample@example.org" \
    -new -key certkey.pem -x509 -days 3660 -outform der -out cert.der
```

The DER files are raw binary, and can be read directly using system open(), read(), and so forth.

To see what a certificate contains, use a command like this:

```
$ openssl x509 -in cert.der -inform der -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 12288164338105855098 (0xaa88545c2a6c2c7a)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: CN=Sample Person/emailAddress=sample@example.org
```

```
        Validity
            Not Before: Nov  5 03:47:09 2014 GMT
            Not After : Nov 12 03:47:09 2024 GMT
        Subject: CN=Sample Person/emailAddress=sample@example.org
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:a9:38:0f:cf:c9:ca:23:18:04:7b:cd:69:02:ba:
                    43:6e:4e:bd:ea:83:4d:72:5b:29:e4:7c:a3:38:aa:
                    ed:de:5f:13:ac:99:b7:4f:73:c9:77:e7:f8:9c:47:
                    f9:c9:65:89:0f:3b:a2:92:c0:b8:ee:0a:9e:a2:ed:
                    81:62:7c:98:ce:be:56:dc:27:6b:ba:be:98:28:07:
                    c0:b9:b6:13:ee:d8:82:ad:97:95:83:91:dc:3d:d3:
                    f4:f2:77:3b:90:6f:15:40:96:c5:a6:cf:9f:9e:05:
                    f7:bb:9a:40:57:ad:19:23:a4:c7:c4:44:f4:f5:b8:
                    4c:94:c0:5e:19:a2:e5:7b:bc:b3:5e:b0:fb:c4:f6:
                    48:29:2c:80:d4:18:4c:95:ff:e7:4c:4a:8b:a7:dd:
                    b1:63:89:89:e7:c4:05:d3:99:e3:9e:cf:78:ce:f6:
                    bd:4d:2d:00:9a:f9:80:fb:a4:63:d5:42:7c:8d:a4:
                    ae:37:e0:1c:a4:91:80:3b:27:11:c4:eb:a2:3d:12:
                    62:39:85:f1:29:31:a5:76:42:44:70:b6:b5:ef:dc:
                    bb:1f:38:2e:e5:03:fd:4a:ea:e7:95:68:e4:52:07:
                    31:c8:bb:ef:86:18:b1:c8:b2:ac:98:25:82:73:b3:
                    c9:2f:ed:36:16:4c:63:f5:24:67:24:1e:71:9b:f1:
                    78:35
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                BA:C1:ED:FF:26:AD:64:99:62:BF:FF:DD:09:B5:60:C0:BB:05:6F:3C
            X509v3 Authority Key Identifier:

keyid:BA:C1:ED:FF:26:AD:64:99:62:BF:FF:DD:09:B5:60:C0:BB:05:6F:3C

            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: sha1WithRSAEncryption
         91:ba:c6:9b:b9:f5:dc:2d:72:ce:65:1e:ad:82:33:af:43:23:
         cc:cf:19:61:04:09:e7:67:f1:68:71:36:4d:14:dc:6d:cd:93:
         18:48:61:b7:14:9a:cc:b8:e6:c7:7e:8f:60:aa:11:d7:f1:5d:
         fb:ba:cc:ae:84:6c:37:8c:3c:a3:5d:2c:85:75:62:35:37:1d:
         9c:d4:ad:23:76:b2:fa:b8:1c:32:00:4b:aa:e5:c6:ce:29:39:
         92:cf:d7:ac:af:c8:94:1c:2b:e3:3c:2e:c1:40:0b:93:ab:02:
```

```
eb:26:c3:92:09:61:d9:7c:08:6d:d2:8f:06:23:bd:d8:21:5d:
b1:e8:2c:f8:82:e8:21:d9:55:6a:c3:25:f1:66:8c:a3:17:0d:
bf:42:26:fe:e2:c8:fe:fb:78:bb:f6:e1:b6:e2:7e:8e:e7:be:
b2:db:ad:e6:c0:d5:56:2f:84:87:7a:2b:70:13:0c:99:70:8a:
fe:f7:ea:4a:20:68:e5:15:70:c7:b3:e7:90:57:ef:ed:cd:40:
fd:7e:54:a2:ae:0b:2b:a3:22:d8:bc:81:b4:99:b7:88:71:5f:
0d:b9:fe:42:ab:e5:b1:48:9b:f0:d7:0e:c6:69:48:1e:c1:eb:
bb:3d:0a:55:95:02:eb:dc:68:52:b0:52:de:48:23:70:ea:03:
91:a6:10:ac
```

Create a DNS record for a certificate

Assuming that the SMIMEA RFC is issued in roughly its current form, the text form of each record has four fields, as described in the next section. The first field is always 3, the second is 0 for a certificate or 1 for a bare key, the third is always 0, and the fourth is the certificate or key in hex. The master file text format allows the fourth field to be broken up into pieces for readability.

The name for the record is formed as described in the next section. To create the certificate using command line tools, first create the SHA-224 hash of the mailbox, in this case “sample”:

```
$ echo -n sample|openssl dgst -sha224
(stdin)= 9003e374bc726550c2c289447fd0533160f875709386dfa377bfd41c
```

Since the domain is example.org, the name will be
`9003e374bc726550c2c289447fd0533160f875709386dfa377bfd41c._smimecert.example.org.`

For the record using the bare key, dump the key in hex:

```
$ hexdump -v -e '1/1 "%02x"' certpub.der
   30820122300d06092a864886f70d01010105000382010f003082010a0282010100a9380
   fcfc9ca2318047bcd6902ba436e4ebdea834d725b29e47ca338aaedde5f13ac99b74f73
   c977e7f89c47f9c965890f3ba292c0b8ee0a9ea2ed81627c98cebe56dc276bbabe98280
   7c0b9b613eed882ad97958391dc3dd3f4f2773b906f154096c5a6cf9f9e05f7bb9a4057
   ad1923a4c7c444f4f5b84c94c05e19a2e57bbcb35eb0fbc4f648292c80d4184c95ffe74
   c4a8ba7ddb1638989e7c405d399e39ecf78cef6bd4d2d009af980fba463d5427c8da4ae
   37e01ca491803b2711c4eba23d12623985f12931a576424470b6b5efdcbb1f382ee503f
   d4aeae79568e4520731c8bbef8618b1c8b2ac98258273b3c92fed36164c63f52467241e
   719bf178350203010001
```

So the full record will be:

```
9003e374bc726550c2c289447fd0533160f875709386dfa377bfd41c._smimecert.exampl
   e.org. SMIMEA (3 1 0
   300820122300d06092a864886f70d01010105000382010f003082010a0282010100a938
   0fcfc9ca2318047bcd6902ba436e4ebdea834d725b29e47ca338aaedde5f13ac99b74f7
```

```
   3c977e7f89c47f9c965890f3ba292c0b8ee0a9ea2ed81627c98cebe56dc276bbabe9828
   07c0b9b613eed882ad97958391dc3dd3f4f2773b906f154096c5a6cf9f9e05f7bb9a405
   7ad1923a4c7c444f4f5b84c94c05e19a2e57bbcb35eb0fbc4f648292c80d4184c95ffe7
   4c4a8ba7ddb1638989e7c405d399e39ecf78cef6bd4d2d009af980fba463d5427c8da4a
   e37e01ca491803b2711c4eba23d12623985f12931a576424470b6b5efdcbb1f382ee503
   fd4aeae79568e4520731c8bbef8618b1c8b2ac98258273b3c92fed36164c63f52467241
   e719bf178350203010001 )
```

For the certificate, the process is the same except that the data is the much larger certificate, and the second field is zero:

```
$ hexdump -v -e '1/1 "%02x"' cert.der
   3082034930820231a003020102020900aa88545c2a6c2c7a300d06092a864886f70d010
   1050500303b3116301406035504030c0d53616d706c6520506572736f6e3121301f0609
   2a864886f70d010901161273616d706c6540657861 6d706c652e6f7267301e170d31343
   1313035303343730395a170d323431313132303343730395a303b3116301406035504
   030c0d53616d706c6520506572736f6e3121301f06092a864886f70d010901161273616
   d706c6540657861 6d706c652e6f726730820122300d06092a864886f70d010101050003
   82010f003082010a0282010100a9380fcfc9ca2318047bcd6902ba436e4ebdea834d725
   b29e47ca338aaedde5f13ac99b74f73c977e7f89c47f9c965890f3ba292c0b8ee0a9ea2
   ed81627c98cebe56dc276bbabe982807c0b9b613eed882ad97958391dc3dd3f4f2773b9
   06f154096c5a6cf9f9e05f7bb9a4057ad1923a4c7c444f4f5b84c94c05e19a2e57bbcb3
   5eb0fbc4f648292c80d4184c95ffe74c4a8ba7ddb1638989e7c405d399e39ecf78cef6b
   d4d2d009af980fba463d5427c8da4ae37e01ca491803b2711c4eba23d12623985f12931
   a576424470b6b5efdcbb1f382ee503fd4aeae79568e4520731c8bbef8618b1c8b2ac982
   58273b3c92fed36164c63f52467241e719bf178350203010001a350304e301d0603551d
   0e04160414bac1edff26ad649962bfffdd09b560c0bb056f3c301f0603551d230418301
   68014bac1edff26ad649962bfffdd09b560c0bb056f3c300c0603551d13040530030101
   ff300d06092a864886f70d0101050500038201010091bac69bb9f5dc2d72ce651ead823
   3af4323cccf19610409e767f16871364d14dc6dcd93184861b7149accb8e6c77e8f60aa
   11d7f15dfbbaccae846c378c3ca35d2c85756235371d9cd4ad2376b2fab81c32004baae
   5c6ce293992cfd7acafc8941c2be33c2ec1400b93ab02eb26c3920961d97c086dd28f06
   23bdd8215db1e82cf882e821d9556ac325f1668ca3170dbf4226fee2c8fefb78bbf6e1b
   6e27e8ee7beb2dbade6c0d5562f84877a2b70130c99708afef7ea4a2068e51570c7b3e7
   9057efedcd40fd7e54a2ae0b2ba322d8bc81b499b788715f0db9fe42abe5b1489bf0d70
   ec669481ec1ebbb3d0a559502ebdc6852b052de482370ea0391a610ac

9003e374bc726550c2c289447fd0533160f875709386dfa377bfd41c._smimecert.exampl
   e.org. SMIMEA (3 0 0
   3082034930820231a003020102020900aa88545c2a6c2c7a300d06092a864886f70d010
   1050500303b3116301406035504030c0d53616d706c6520506572736f6e3121301f0609
   2a864886f70d010901161273616d706c6540657861 6d706c652e6f7267301e170d31343
   1313035303343730395a170d323431313132303343730395a303b3116301406035504
   030c0d53616d706c6520506572736f6e3121301f06092a864886f70d010901161273616
```

```
   d706c65406578616d706c652e6f726730820122300d06092a864886f70d010101050003
   82010f003082010a0282010100a9380fcfc9ca2318047bcd6902ba436e4ebdea834d725
   b29e47ca338aaedde5f13ac99b74f73c977e7f89c47f9c965890f3ba292c0b8ee0a9ea2
   ed81627c98cebe56dc276bbabe982807c0b9b613eed882ad97958391dc3dd3f4f2773b9
   06f154096c5a6cf9f9e05f7bb9a4057ad1923a4c7c444f4f5b84c94c05e19a2e57bbcb3
   5eb0fbc4f648292c80d4184c95ffe74c4a8ba7ddb1638989e7c405d399e39ecf78cef6b
   d4d2d009af980fba463d5427c8da4ae37e01ca491803b2711c4eba23d12623985f12931
   a576424470b6b5efdcbb1f382ee503fd4aeae79568e4520731c8bbef8618b1c8b2ac982
   58273b3c92fed36164c63f52467241e719bf178350203010001a350304e301d0603551d
   0e04160414bac1edff26ad649962bfffdd09b560c0bb056f3c301f0603551d230418301
   68014bac1edff26ad649962bfffdd09b560c0bb056f3c300c0603551d13040530030101
   ff300d06092a864886f70d0101050500038201010091bac69bb9f5dc2d72ce651ead823
   3af4323cccf19610409e767f16871364d14dc6dcd93184861b7149accb8e6c77e8f60aa
   11d7f15dfbbaccae846c378c3ca35d2c85756235371d9cd4ad2376b2fab81c32004baae
   5c6ce293992cfd7acafc8941c2be33c2ec1400b93ab02eb26c3920961d97c086dd28f06
   23bdd8215db1e82cf882e821d9556ac325f1668ca3170dbf4226fee2c8fefb78bbf6e1b
   6e27e8ee7beb2dbade6c0d5562f84877a2b70130c99708afef7ea4a2068e51570c7b3e7
   9057efedcd40fd7e54a2ae0b2ba322d8bc81b499b788715f0db9fe42abe5b1489bf0d70
   ec669481ec1ebbb3d0a559502ebdc6852b052de482370ea0391a610ac )
```

These records can be placed in the zone's master file.

If the DNS server does not yet handle SMIMEA record syntax, the records can still be added with slightly more difficulty using the generic TYPEnn syntax described in RFC 3597, which requires the specification to have a byte length followed by hex digits with each pair of digits describing a byte of the record. Each of the first three fields in the record is a single byte. The public key in the example above is is 294 bytes long, so the data for the record containing it is a total of 297 bytes. Assuming the type number is private type 65444, the record would be:

```
9003e374bc726550c2c289447fd0533160f875709386dfa377bfd41c._smimecert.exampl
   e.org. TYPE65444 \# 297 (03 01 00
   300820122300d06092a864886f70d0101010500038201 0f003082010a0282010100a938
   0fcfc9ca2318047bcd6902ba436e4ebdea834d725b29e47ca338aaedde5f13ac99b74f7
   3c977e7f89c47f9c965890f3ba292c0b8ee0a9ea2ed81627c98cebe56dc276bbabe9828
   07c0b9b613eed882ad97958391dc3dd3f4f2773b906f154096c5a6cf9f9e05f7bb9a405
   7ad1923a4c7c444f4f5b84c94c05e19a2e57bbcb35eb0fbc4f648292c80d4184c95ffe7
   4c4a8ba7ddb1638989e7c405d399e39ecf78cef6bd4d2d009af980fba463d5427c8da4a
   e37e01ca491803b2711c4eba23d12623985f12931a576424470b6b5efdcbb1f382ee503
   fd4aeae79568e4520731c8bbef8618b1c8b2ac98258273b3c92fed36164c63f52467241
   e719bf178350203010001)
```

The white space added to the the hex string for readability is ignored.

Retrieve a certificate from the DNS

To retrieve a certificate from the DNS

1. Map the email address into a domain name:

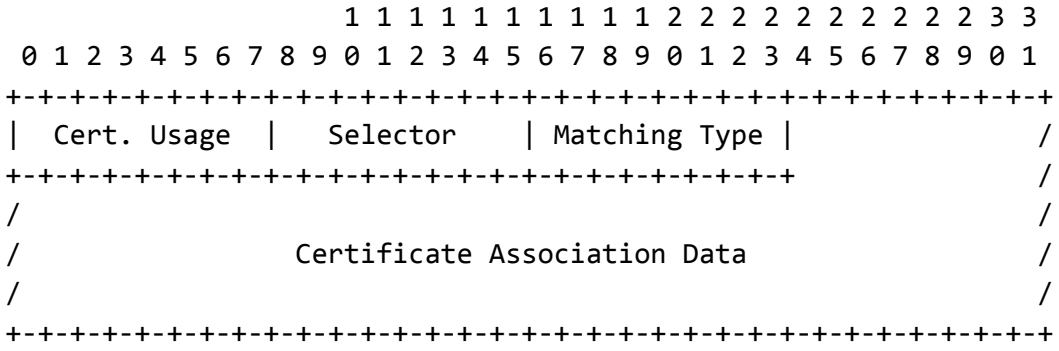
Separate the address into the mailbox, the part before the @ sign, and the domain, the part after the @-sign. Take the mailbox part of the address, and create its SHA-224 hash. (In Openssl, use the EVP_Digest functions.) Represent that hash as a string of hexadecimal digits.

Then create a domain name of the hash string, followed by the string `._smimecert.` followed by the e-mail address' domain name. For example, if the email address were robert@bigbank.com, the hash of "robert" is ecb453d83da2067efeb1243964d2b00aca62a7230c64e39ce69ac555 so the domain name to look up would be `ecb453d83da2067efeb1243964d2b00aca62a7230c64e39ce69ac555._smimecert.bigbank.com`.

2. Look up the SMIMEA record(s) under that name using the getdns call getdns_general() with the dnssec_return_only_secure extension. Note that at this time, there is no resource record type for SMIMEA, and one has not yet been applied for, so all queries and responses need to use private type numbers from the range 65280-65534 until a public record type is assigned.
3. Check that the return value is **GETDNS_RETURN_GOOD** and the response parameter contains **GETDNS_RESPSTATUS_GOOD**, meaning that at least one record was found.
4. In the response, the "replies_tree" entry is a list of the SMIMEA records that were found and were DNSSEC validated. If the list is empty, that means that either there were no SMIMEA records, or that none that were found were DNSSEC validated.
5. For each entry in the "replies_tree", there is an "answer" entry that is a list of one or more answer records. In each record, there is a "rdata" entry, each of which has an "rdata_raw" entry. That lowest-level rdata_raw entry contains the RDATA field that contains the certificate.

Note that according to e-mail standards (RFCs 5321 and 5322) the mailbox name is case sensitive, so `robert@example.com`, `Robert@example.com`, and `ROBERT@example.com` are different addresses. The hash value depends on the exact characters in the mailbox, so those three addresses would have different hash values, and hence different names in the DNS. Although it is a common convention to treat all case variations of a mailbox the same, there's no standard way to ensure that. Heuristic approaches can help, such as trying an all lower case or all upper case version of a mailbox if the original version isn't found.

The RDATA for each SMIMEA record has this format:



The first three bytes are called Certificate usage, Selector, and Matching Type. The format of the rest of the RDATA depends on the first three bytes, but in this application is always a certificate.

5. Check that the Certificate Usage byte contains the value 3, meaning that the record contains a certificate issued by the domain. If it contains any other value, disregard this SMIMEA record.
6. Check that the Selector field contains 0, meaning that the record contains a certificate rather than only a public key. If it contains any other value, disregard this SMIMEA record.
7. Check that the Matching Type field contains 0, meaning that the record contains an actual certificate rather than a verification hash. If it contains any other value, disregard this SMIMEA record.
8. The Certificate Association Data is a certificate for the requested e-mail address.

Display S/MIME indicators in message list

It may be useful to show an indicator of whether a message was signed or encrypted in the MUA's message list. If so, each message's `Content-Type:` header provides a fairly reliable way to identified signed or encrypted messages.

If the Content-Type: is "application/pkcs7-mime", the message is usually encrypted. If the Content-Type: is “multipart/signed”, the message is signed.

Occasionally messages are partially signed, e.g., a user sends a signed message to a mailing list which adds an unsigned footer. In that case, the message’s Content-Type: will be multipart/related, with one of the enclosed parts being multipart/signed. Existing MUAs are quite inconsistent at recognizing signed or encrypted sub-parts, and users do not generally expect partially signed or partially encrypted messages to work.

Check the signature on incoming message

1. Retrieve the From: address as described above.
2. Extract the signed part of the message, which is “Content-Type: multipart/signed” and typically includes two subparts, one for the signature and one for the signed text.
3. Fetch the certificates in the SMIMEA record(s) for that address as described above. If none are found, the signature is invalid. For each certificate found, pass the certificate and the signed part of the message to the S/MIME validation routine. (In Openssl, call CMS_verify(). Be sure to set the CMS_NOINTERN flag to tell it not to look for additional certificates in the signature block.)
4. If the verification routine succeeds, it will return the part of the message that was signed. Display that, along with an indication that the signature was valid.
5. If none of the validations succeed, the signature is invalid.
6. If the signature is invalid, extract the signed text subpart from the signed part of the message, and display it with an indication that the signature was invalid.

Decrypt an incoming message

1. Retrieve user's private key(s) from local storage. Retrieve the user's own certificate if it is not locally stored. If the user has multiple addresses and multiple private keys, retrieve the certificate for each address for which there is a private key.
2. Extract the encrypted part of the message, which is "Content-Type: application/pkcs7-mime".
3. For each private key, pass the private key, the corresponding certificate, and the encrypted part of the message to the decryption routine. (In Openssl, this is CMS_decrypt().)
4. If the decryption succeeds, the routine will provide the plaintext version of the encrypted message part. Check the Content-Type of the plaintext to see if it is multipart/signed. If so, the message is both encrypted and signed.
5. If the decryption succeeded and the plaintext is not signed, display the plaintext with an indication that the message as encrypted.
6. If the decryption succeeded and the plaintext is signed, check the signature on the plaintext as described above, and display the signed part of the plaintext with an indication that the message was encrypted, and the signature was or was not valid, as appropriate.
7. If all decryption attempts failed, display a synthesized message body that says that the message could not be decrypted.

Sign an outgoing message

1. Retrieve the user's private key(s) from local storage. Retrieve the user's own certificate if it is not locally stored. If the user has multiple addresses and multiple private keys, retrieve the address on the From: line, and use the private key and certificate for that address.
2. Retrieve the body of the message. It will typically have a Content-Type of multipart/alternative, multipart/related, text/plain, or text/html, but any content type can be signed. If the body has no MIME headers, treat it as text/plain.
3. Pass the message body, the private key, and the certificate to the signing routine. (In Openssl, this is CMS_sign(). Set the CMS_NOCERTS flag so it does not include any certificates in the signature block.)
4. The signing routine will return a MIME multipart/signed block of text. If the message is to be both signed and encrypted, encrypt the signed block as described below.
5. If the message is not to be encrypted, insert the signed block into the message in place of the existing body, and send the message.

Encrypt an outgoing message

1. Extract the recipient address(es) from the message's To: and Cc: headers.
2. Fetch a certificate in the SMIMEA record for each addresses as described above. If none are found for an address, the message cannot be encrypted for that recipient. The user must either remove the recipient, or send the message unencrypted. If a recipient has multiple certificates,

choose one arbitrarily.

3. Retrieve the body of the message. It will typically have a Content-Type of multipart/alternative, multipart/related, text/plain, or text/html, but any content type can be encrypted. If the body has no MIME headers, treat it as text/plain. If the body has already been signed as described above, the body will be of type multipart/signed.

4. Pass the list of recipient certificates and the message body to the encryption routine. (In Openssl, this is CMS_encrypt().)

5. The encrypting routine will return a MIME application/x-pkcs7-mime block of text. Insert the signed block into the message in place of the existing body, and send the message.