SMIMEA DANE Secure Message Test Plan

PRELIMINARY DRAFT

These tests are intended to exercise the features of the SMIMEA DANE extension. Many examples use two users: S sends messages, and R receives messages. Some examples use multiple recipients R2, R3, and so on.

Required test scenarios and dimensions

The security of this design depends on the cryptographic security of DNS messages protected by DNSSEC, and e-mail message bodies protected by S/MIME. In both cases, the signatures are in the messages themselves, and do not depend on the security of the channels through which the messages are transferred, which makes the testing somewhat simpler.

The test process involves creating cryptographically valid and invalid messages of each type, and checking that the valid ones pass verification and the invalid ones do not. In most cases, the verification information is public, that is, anyone can verify that a DNSSEC signature is valid. In some cases, the verification information is private, such as the private key to decode an encrypted message. In the latter case, tests need to check both that a person with the key can verify and decode the message, and a person without the key cannot.

Set up of test cases

The test data generally resides in two places: DNS data in DNS servers, and e-mail messages in message stores. DNS servers generally retrieve data from “master files” that are maintained manually or semi-automatically, and serve it to DNS clients. Mail clients, often called Mail User Agents or MUAs, retrieve incoming messages from message stores on a mail server using schemes called POP or IMAP. They send out mail that is created by a user using SMTP.

Creating valid test data is straightforward: use the software to create keys and messages using the extension. Creating invalid test data is harder, but in the context of this testing, not greatly so. Rather than modify DNS servers or MUAs to create invalid data, it is sufficient to modify good data to be invalid. If one edits the zone file a DNS server uses to invalidate or remove a DNSSEC signature, the server will then serve the invalid data. Similarly, if one edits a mail message in the message store to invalidate a signature or a certificate, the MUA will retrieve the invalidated message. Since neither the zone file nor the message store are part of the security system, these techniques are in practice equivalent to modifying the software, while being much easier to implement.

Means of validation

There are two separate validation approaches, *internal* and *external*.

Internal validation uses the software itself. For example, a test might involve receiving a signed message from a known valid signer. The software should report that it received the message and the signature was valid. External validation uses separate tools to check the validity of data. A wide variety of open source tools are available for external validation.

For DNSSEC validation, popular tools such as “DNSSEC-Tools” or OARC’s “ODVR”include a tool to fetch records from the DNS and check that the DNSSEC signatures are valid.

For S/MIME validation, the widely available openssl package contains extensive tools to check and report on the validity of S/MIME messages relative to user-supplied keys. Other similar tools are available such as Gnu Privacy Guard’s GPGSM to handle S/MIME messages.

Security testing

The security guarantees of the extension are that it will correctly encrypt and sign outgoing messages, and correctly decrypt and validate incoming messages. Those guarantees are easy to test as part of the required test scenarios, but could lead to a false sense of security if the system is encoding and decoding its own bugs. Therefore, the test harness should use an S/MIME engine that has a different origin than the one being used in the extension.

The extension can only introduce new vulnerabilities in MUAs or webmail systems that already do S/MIME, and even there, the only new vulnerabilities are by vouching for certificates that are in fact not validated.

Test plan

Below is a sketch of the tests that will validate that the software operates correctly. The tests are divided into three sections, for key management, signatures, and encryption.

Key management

1. Create a certificate for S in an MUA. Upload it to the DNS server. Verify that the certificate appears in the DNS, and that its public key matches the private key in the MUA.
2. Tell the DNS server to remove the certificate for S. Verify that the key no longer appears in the DNS. (Wait TTL to allow cache to flush, or test in a way that bypasses the DNS cache.) In subsequent tests, this is called the *S removed key*.
3. Create and install another certificate for S and install it into the DNS. Edit the DNS zone file to make the DNSSEC signature invalid by changing the enclosing RRsig for the record.
    a. Check that the key can no longer be retrieved.
    b. Remove the signature completely, and again check that the key can longer be retrieved.
    c. This key is called the *S invalid key*.
4. Create and publish certificates for S and R and R2 for subsequent tests. These are called the S and R and R2 *active keys*.

Signed messages

4. Create a message from S to R, sign it with the S active key, send to R.
    a. Check that R receives the message.
    b. Check that R can extract the identity of S from the message.
    c. Check that R can retrieve S’s active key.
    d. Check that R can verify the signature with the key.
5. Create a message from S to R, sign it with the S removed key, send to R.
    a. Check that R receives the message.
    b. Check that R can extract the identity of S from the message.
    c. Check that R can *not* retrieve S’s removed certificate.
    d. Check that R cannot verify the signature.
6. Create a message from S to R, sign it with the S invalid key, send to R.
    a. Check that R receives the message.
    b. Check that R can extract the identity of S from the message.
    c. Check that R can *not* retrieve S’s invalid key.
    d. Check that R cannot verify the signature.
7. Create a message from S to R, sign it with the S active key, send to R. Add or delete a few characters of the signed message, by changing it in the message store.
    a. Check that R receives the message.
    b. Check that R can extract the identity of S from the message.
    c. Check that R can retrieve S’s active key.
    d. Check that R can *not* verify the signature with the key, and the MUA reports failure.

Encrypted messages

8. Create a message from S to R, encrypt it with the R active key, send to R.
    a. Check that S can retrieve R’s key.
    b. Check that S can encrypt the message,
    c. Check that R receives the message.

    d. Check that R can decode the message with the key.

9. Remail the message from the previous example to R2

    a. Check that R2 can *not* decode it, and the MUA gives an appropriate error message.

10. Create a message from S to R and R2, encrypt it with the R and R2 active keys, send to R.

    a. Check that S can retrieve R and R2's key.

    b. Check that S can encrypt the message,

    c. Check that R receives the message.

    d. Check that R can decode the message with the key.

    e. Check that R2 receives the message.

    f. Check that R2 can decode the message with the key.

11. Create a message from S to R, encrypt it with the R active key, send to R. Add or delete a few characters of the encrypted message, by changing it in the message store.

    a. Check that R receives the message.

    b. Check that R can *not* decode the message, and the MUA reports failure

12. Create a message from S to R and R2, encrypt it with the R and R2 active keys, send to R and R2. Add or delete a few characters of the encrypted message, by changing it in the message store.

    a. Check that R and R2 receive the message.

    b. Check that R and R2 can *not* decode the message, and the MUA reports failure.