DAME SMIMEA Developers Background Guide

Introduction

E-mail is one of the oldest, yet still one of the most important services on the Internet. The service was formally defined in the 1980s, in RFC 821 which describes SMTP, the system used to transfer a message from one computer to another, and RFC 822 which describes the message format. In the subsequent 30 years, the mail system has been vastly expanded, while remaining compatible with older software. In particular, most new features such as attachments, formatted text, and cryptographic security have been added via extensions to the message format, without needing changes to SMTP or existing SMTP software.

Internet e-mail was designed and implemented in an environment where users were considered trustworthy, many knew each other personally, and the rare security issue was handled by talking to the people involved. As the Internet has grown, most users are strangers to each other, and even when two people know each other, the network between them is usually operated by strangers.

An e-mail message has two main parts: the headers and the body. Headers contain information about the message, such as where it is supposed to be delivered (the "To:" field), when it was sent (the "Date:" field), a hint about the contents (the "Subject:" field), and so on. The message itself comes after the headers.

Secure e-mail provides two general features, *signing* and *encryption*. When a message is signed, the sender attaches his or her cryptographic identifier to it in a tamper-resistant way. When a recipient receives a message with a valid signature, that means both that the message really is from the sender who signed it, and that it hasn't been changed at all in transit. An encrypted message is intended for a specific recipient or set of recipients, and can only be decoded by those recipients. If the message is intercepted by someone else, its contents are unreadable. Most encrypted messages are also signed.

Overview of Message Security

Overview of Cryptography

Modern cryptography consists of many types of operations that can be combined to give protection to messages (information that is at rest, such as on disk or in memory) and to communication (the act of two or more systems contacting each other and passing messages). Each type of operation imparts a different type of protection and is based on different security assumptions.

In the descriptions below, only two parties are involved, even though in theory many of them work with multi-party security.

One of the simplest to understand forms of cryptography is symmetric encryption. In symmetric encryption, both parties know a key that is unknown and unguessable by anyone else. The key is used to transform a message from its readable form to an unreadable form; the same key is used to transform the unreadable form back to the readable form.

A message in its unreadable form is often likened to the message being in a sealed envelope. You can pass the sealed envelope around and no one can see the actual message. In this analogy, the key is used to seal and unseal the envelope; only those who know the key can do the sealing and unsealing.

There are many different algorithms that can be used to perform symmetric encryption. Today, the most common is the Advanced Encryption Standard (AES), developed as part of an international multi-year competition organized by NIST.

Asymmetric encryption is quite different than symmetric encryption in that two keys are used when transforming a message to and from its unreadable form. The first key, called the *private key*, is known by only the one person who creates the two keys at the same time; the second key, called the *public key*, can be known by anyone. The two keys together are called a *key pair*.

There are two primary uses for asymmetric encryption: digital signatures and key exchange. A digital signature is created when the private key is used to make a message unreadable. Anyone who has the public key can be sure that only the person who has the private key created the unreadable message; thus, the private key becomes like an unforgeable signature. In key exchange, someone uses the public key to create a message that can only be made readable by the private key; this allows anyone to make a message that only the one person with the private key can read. Using asymmetric keys for key exchange means that the exchange can happen in public instead of requiring the two parties to somehow secretly pass the encryption key to each other.

There are multiple types of algorithms that can be used in asymmetric cryptography. The most common type is called *RSA*, which are the initials of its inventors. Nearly all signed and encrypted e-mail today uses RSA. A second type is called *EC*, which stands for "elliptic curve", which is the type of math used in the cryptographic calculations. RSA keys are much larger than EC keys for the same strength of encryption, which is why many people hope that EC catches on more in the future.

Certificates and Keys for Secure E-mail

A *certificate* is a copy of a public key that is signed with the public key of a trusted entity and contains an identifier for the person who knows the private key in the key pair. For e-mail certificates, the identifier is the person's e-mail address. Thus, an e-mail certificate is a signed statement saying "the private key associated with this public key is known by the person who sends and receives mail at this address".

The entity that creates certificates is called a *certificate authority* (*CA*). A CA is an organization that has been trusted to verify that a particular person is in possession of a particular private key. By trusting a CA, you trust that any messages that they sign with that private key can be assumed to be associated with that person. CAs sign the certificates they create so that the assertions in the certificates cannot be changed by malicious parties.

For example, if someone who sends e-mail can prove to a CA that their public key has a particular value, other people might trust that CA to do the validation of that assertion. At that point, those people will trust the identity of the sender on any e-mail they receive that has a signature validated by the CA.

When you receive a piece of signed e-mail, you need to find the certificate associated with the address of the e-mail. Similarly, when you want to send a piece of encrypted mail, you need to know the recipient's

certificate so you can find their public key. There are many ways that this can be done, although no particular mechanism is common enough to be considered "the" way to do it. There are some directories with e-mail certificates, but they don't have many certificates in them and they thus not heavily used. Some mail programs include certificates in the messages they send, but not all do. The task of matching is made more difficult by the fact that there are many different places in a certificate that might have the associated e-mail address, so CAs often put the same e-mail address in a certificate in multiple places.

Self-issued certificates use the same key for both the signer and subject. These certificates don't provide any CA assurance, but can contain other material such as an identity (in this case, the e-mail address). If the source of the certificate is trusted, as described below, the certificate itself is considered trustworthy.

There is an emerging new standard for publishing certificates in an easy-to-find place, namely in the DNS. In this new system, the certificate for bob@example.com would be found by looking for it in the DNS in the same place that you look for the address of the example.com mail server. The standard, and a way to automate its use, is described later in this document.

Overview of S/MIME

How S/MIME uses MIME

When e-mail was first introduced, a "message" was just one block of plain text. It rapidly became clear that a message should be able to more than just text (such as instead being a picture or a music recording), and also that a message should be able hold many types of content at the same time (such as a magazine page that has headings, pictures, captions, and so on).

The method for making a single mail message be able to hold many things at once is called *MIME*. A MIME message is a container for parts, and each part has a label on it saying what kind of content the part has. (Message parts are sometimes called "attachments".) MIME was quickly adopted by nearly all mail software.

Later, people became more concerned with security, MIME was extended to allow message parts, or the entire message, to be encrypted, signed, or both. An encrypted message can only be read by the person who has the private key in the key that was used for encrypting. A signed message allows the recipient to be sure that the person who claims to have sent it is really the person who did. The S/MIME standard leverages MIME as a format so that recipients of signed messages can still read the message even if the recipient doesn't use S/MIME.

Capabilities of S/MIME

S/MIME allows mail messages to be encrypted, signed, or both signed and encrypted. It uses standard cryptographic encryption functions such as AES to change the message parts into their encrypted equivalents, and signature functions such as RSA and ECDSA to sign the message parts to assure their authenticity and integrity. A message with multiple parts can have all or just some of the parts protected by S/MIME, although it is most common to have entire messages protected. Note, however, that a message's header is not signed or encrypted: only the message is.

The S/MIME protocol leverages the universally-deployed MIME format to specify to a recipient which part of a message is protected and which cryptographic algorithm was used for the protection by the message's author.

Most major mail user agents (MUAs) have had S/MIME support for many years. The MUAs that come for free with Windows (Windows Live Mail), Mac OS and iOS (Apple Mail), and most versions of Linux (Mozilla Thunderbird) have built-in S/MIME support. The MUA for Android phones and tablets do not, but S/MIME plugins such as djizgo (from http://djigzo.com/android.html) are available. Many users get their mail with webmail systems such as Gmail or Yahoo mail; very few of these systems natively support S/MIME. MUAs that do not have S/MIME support almost always have existing extensions to add all S/MIME capabilities.

Finding Certificates for S/MIME

One of the most noticeable limitations of S/MIME is that, when you want to send an encrypted message to someone from you have never received mail, you will not know the public key you are supposed to use for encrypting. If you have already received a message from them, and they included their certificate, that certificate includes their public key, and a signature from a CA; if you trust that CA, you can encrypt a message to them. Users in enterprises that have an LDAP directory server might be able to use that server to find another user's PKIX certificate. Beyond these two circumstances, however, there is currently no standard way to find someone's public key, and without that, you cannot safely encrypt a message for them.

Similarly, if you get a signed message from someone, the message might not include the signer's certificate (because sending that in every signed message makes the message larger), or they may have included a certificate that is signed by a CA you do not trust. There is currently no way to find an acceptable signing certificate for them if it was not included in the signed message.

S/MIME Signing and Encrypting

S/MIME allows individual parts of a message or the whole message to be signed or encrypted. In almost all MUAs today, the entire message is protected at once. In fact, few MUAs allow you to specify which individual parts would be protected. Note that only the message is protected: the headers (the "To:", "From:", "Subject:", and so on) are not protected at all.

There are two ways for S/MIME to format signed messages, using different MIME formatting. The first, called "multipart/signed", allows receivers whose MUAs do not support S/MIME to read the message; the second, called "application/pkcs7-mime", can only be read in MUAs that support S/MIME. The reason an MUA would use the second form is that some old mail servers will reformat messages in the first format, which then makes the signature invalid. This remains one of the most challenging issues for S/MIME deployment.

Encrypting in S/MIME is done in exactly one way, using "multipart/encrypted". Because encrypting is meant to make the message unreadable to anyone without the corresponding key, MUAs that do not support S/MIME cannot interpret the contents of a multipart/encrypted message.

Sending Signed and Encrypted Messages

An MUA almost always lets you decide whether or not to add S/MIME protection to outgoing messages; this is often done on a message-by-message basis. Typically, an MUA that is S/MIME-capable will have two button in the window for new messages, one for signing and one for encrypting. For example, one popular program shows the following before S/MIME has been applied:

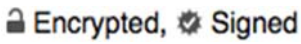
When encryption and signing are applied, the buttons turn to:

When an MUA is told to send a signed message, the MUA retrieves your public key from local storage, asks you the password for the key (if there is one), does the necessary cryptography on the message, and adds the additional S/MIME sections to the message. When an MUA is told to send an encrypted message, the MUA retrieves the public key of the recipient, does the necessary cryptography on the message, removes the cleartext of the message, and adds the additional S/MIME sections to the message.

It is common to want to both sign and encrypt a message. The MUA can decide whether to sign first then encrypt, or encrypt first then sign. There are advantages to both choices: sign-then-encrypt protects the data in the signature from being seen by anyone during transit, but encrypt-then-sign assures the recipient that the sender was the one who encrypted the message. There is no widespread agreement in MUAs about which method is better.

Receiving Signed and Encrypted Messages

The S/MIME standard does not cover how signed and encrypted messages should look in an MUA. Different systems show different icons for when a received message is encrypted or signed. Some MUAs show the indicators at the top of the message, some show it in the “From:” field. For example:



The steps an MUA takes when it receives a signed or encrypted message are similar to those it takes when it sends such a message, but in reverse. When it receives a signed message, the MUA retrieves the public key of the recipient, then validates that the message is untampered by whether or not the digest of the message that was sent matches the digest of what was received. When an MUA receives an encrypted message, it retrieves your public key from local storage, asks you the password for the key (if there is one), then does the decryption.

Storing Signed and Encrypted Messages

After an MUA has opened a signed message, it can store the message in the user’s storage the same way it normally does all other messages. However, there is the question of whether or not to store the signature with the message. Long-term storage of signatures is problematic because the signer’s certificate might

expire, the message might get changed in storage (and thus invalidate the signature), and so on. Many MUAs store signed messages with an indication that the signature was validated when the message was received, but without the actual signature.

Storing encrypted messages is a more vexing problem. The main question is whether to store those messages encrypted (thereby preserving their protection, but making the message unsearchable) or decrypted (making the message searchable but with less protection than when it was sent). There is no widespread agreement on how to store messages that were encrypted in transit.

Forthcoming Capabilities for S/MIME Certificate Retrieval

The main gap in S/MIME as a certificate management system is the ability to find certificates for someone you don't already have the correct certificate for. There have been different proposals for certificate servers for nearly 20 years, some based on the idea of a universal X.500-based directory, others based on private services that people would subscribe to. None of the proposed solutions have gotten significant traction in the market, however.

In 2012, the IETF standardized a way for TLS clients to find key and certificate information about TLS servers through the DNS known as DANE. That work is slowly being deployed for many types of servers that use the TLS protocol. The IETF is currently working on using a very similar technology to allow S/MIME users to have similar capabilities as TLS clients using a new DNS record called "SMIMEA". The proposal is embodied in a pre-standard Internet Draft called "Using Secure DNS to Associate Certificates with Domain Names For S/MIME"; the draft is identified by its file name, "draft-ietf-dane-smime".

After the proposal is standardized, an S/MIME user who wants to know the keys or certificates used by an e-mail correspondent will be able to look for those in the DNS. In order to do that, they need to do a DNS lookup for one or more SMIMEA records based on the correspondent's e-mail address; the records have to be validated by the DNSSEC protocol before they are given to the user. There is no assurance to the user that the correspondent will have SMIMEA records in the DNS, and adoption of the forthcoming standard is expected to be slow. However, if they one or more records are there, the user can be assured that the correspondent's DNS administrator put the SMIMEA records in the DNS and thus have confidence that they are valid.

The SMIMEA records can contain the correspondent's PKIX certificate; in this sense, DANE is being used as a certificate directory. The SMIMEA record can instead contain the correspondent's bare key (not in a PKIX certificate) or self-issued certificate, and the requestor can use that key without having to trust any PKIX CA because they know that the key or certificate is associated with the correspondent by the correspondent's DNS administrator. Different organizations will have different policies about how to trust certificates or bare keys from DANE.

In order to use the SMIMEA record to discover a correspondent's certificate or key, an MUA or extension needs to know the e-mail address of the correspondent. It then converts this into a DNS query. When the query comes back, the agent checks the DNSSEC status of the query and throws away any that are not DNSSEC valid. For each SMIMEA record, it extracts either the certificate or keying material, as described in the DANE specification.

In order for a user to publish their key or certificate in the DNS, he needs to communicate with the DNS administrator; this communication mechanism is not standardized. The DNS administrator converts that into an SMIMEA record, inserts the record into the DNS zone that matches the user's e-mail address, and uses standard DNSSEC mechanisms to re-sign the DNS zone.

Extending MUAs and Webmail to Support S/MIME with DANE

The following is a new design for allowing much greater use of S/MIME while requiring little action on the part of the user. It does not change any of the current technologies of S/MIME, PKIX, DNSSEC, or DANE; instead, it combines them in in MUAs so that users can encrypt and sign messages without having to know almost anything about the technologies involved.

In this design:

- The user instructs their MUA to generate a new self-issued S/MIME certificate for signing and encrypting. The user tells the DNS administrator of their site about their new certificate, and the DNS administrator creates a DANE record for it.
- Any time the user sends e-mail to another user, the MUA looks for that other user's DANE record(s) and, if one or more is found, uses the certificates for encrypting the outgoing message. Regardless of whether the message is encrypted, it is also signed by the sender's public key.
- Any time the user receives a message that is signed by S/MIME, the MUA looks for that other user's DANE record(s) and, if found, uses one of those certificates for validating the signature. Any encrypted messages are automatically decrypted.
- If the user loses their private key for a certificate, they simply instruct the DNS administrator to remove the DANE record for that certificate. If the user loses the private key for a CA-issued certificate, they also need to instruct the CA to revoke the certificate.
- If the user has MUAs on different devices (such as one on their desktop computer and one on their phone), each MUA gets its own certificate. The DNS administrator adds each device's certificates to the DNS.

A user whose domain is DNSSEC protected can send signed messages and receive encrypted messages without having to have any interaction with a CA. The extension allows them to create self-issued certificates that reside in the DNS; those certificates can be accessed and used by anyone, not just users of the same extension.

The extension allows MUA or webmail users to sign and encrypt messages with minimal setup. When mail is received, the extension automatically validates any S/MIME signatures and decrypts encrypted messages. Even if the e-mail user does not have their own certificate (such as if their domain name is not DNSSEC protected), the extension will validate incoming signatures and attempt to encrypt outgoing messages.

Requirements

The extension needs to be able to make DNS requests, to validate that the chain of responses all have the required DNSSEC signatures, and to extract certificates and keys from the SMIMEA records in the responses. Note that some browsers prohibit the use of extensions that are able to make DNS requests, so

the extension cannot work in those browsers. There will possibly be a new protocol in the future that will work around this problem.

If an MUA or webmail site does not support S/MIME, the extension needs to add those capabilities in addition to the DANE capabilities. S/MIME extensions are significantly more complex because they need to interact with many different parts of the MUA or browser. For MUAs, the extension has to alter the GUI in two places: in the message creation window (to add buttons for signing and encrypting), and in the incoming message window (to show signature or encryption status). For webmail, the extension has to modify the Javascript that comes from the webmail site to intercept outgoing and incoming mails.

If an MUA or webmail site already supports S/MIME, the DANE extension only needs to be able to affect the MUA's certificate discovery. That is, the value of the DANE extension will be to allow the existing S/MIME engine to add certificates associated with a correspondent, and to validate certificates found in incoming mail. When the extension sees that an incoming message is signed with S/MIME, and the MUA doesn't already have the sender's certificates, the plugin would attempt to retrieve the sender's certificates in the DNS using DANE before the signature validation is done. Conversely, if an outgoing message is to be encrypted, and MUA does not already have the recipient's certificate, the plugin would look up the recipient's certificate in the DNS using DANE before encrypting.

The extension has to be tailored to each environment it runs in. Different MUAs have different requirements for how extensions can interact with them. When the extension is being used for webmail, it can only be used with particular webmail sites that allow extensions to modify their behavior. It is expected that many common MUAs and large webmail sites will work with such an extension.

Processing SMIMEA Records

When the extension is given an e-mail address, it uses that address to search for one or more SMIMEA records. For each SMIMEA record found, the DNSSEC protection for the SMIME record is validated. If it is not valid, the record is treated as if it was not there.

As with any DNS queries, a single SMIME query might have multiple records in the Answers section of the response. This would indicate that the user has multiple certificates and keys. For example, there might be different signing and encrypting keys, or the user might use multiple algorithms (such as having both RSA and ECDSA signing certificates).

SMIMEA records can contain either certificates or keys. The encoding of the RDATA field in the SMIMEA record is described fully in section 2.1 of RFC 6698. For each of the valid records, the extension uses the certificate or key from the record. Software libraries for handling signed or have better support for certificates than for keys, so the subsequent discussion assumes the use of certificates.

An application that is adding support for using certificates from SMIMEA records would use the following steps for retrieving and using the keying material. These steps are based on the current specification in draft-ietf-dane-smime-07; they could change if discussions in the WG change the protocol.

To retrieve a certificate for a user with a given email address, the application processes the address as specified in Section 3 of draft-ietf-dane-smime-07. The result would be a DNS query that would have a

host name that looks like "3f51f4663b2b798560c5b9e16d6069a28727f62518c3a1b3._smimecert.example.com". The application then uses a DNS query mechanism to send a query for that name in class IN with type SMIME to a recursive resolver; this mechanism might be in the operating system (such as if it has the getdns API installed) or in the application itself.

Retrieving SMIMEA records requires DNSSEC. If the application or operating system is relying on an external recursive resolver, two things are mandatory:

- The recursive resolver must do DNSSEC validation on all results
- The operating system must trust the integrity of its communications with the recursive resolver

A different approach that does not assume a validating recursive resolver would be to do validated recursive resolution in the operating system itself, such as with the getdns API or some other DNSSEC-aware API.

Once the SMIME record or records are received by the application, it must extract the certificate needed for processing from the RDATA of the response. The steps are:

1. Check that the type of certificate usage (the first octet of the RDATA) is type 3. If it has a different usage type, fail.
2. Verify that the entire certificate was delivered; that is, verify that the matching type field (the third octet of the RDATA) is 0. If it has a different matching type field, fail.
3. If the selector field (the second octet of the RDATA) is 0, verify that the certificate (the remainder of the RDATA) indicates the correct usage for the public key (signature or encryption). This check is not needed if the selector field is 1.
4. Use the certificate from the record.

The resulting certificate is then used for validating signatures and encrypting messages for the given email address.

If the extension is validating the signature of an incoming S/MIME message, it sees if any of the keys are the one in S/MIME message; if so, it uses that key to do the S/MIME validation. If the extension is creating an encrypted message, it encrypts the message, then encrypts the encryption key with each of the keys extracted from the SMIMEA records, and includes each of those in the message it creates.

Entering and Maintaining SMIMEA Records

The DNS administrator need to be able to receive new keys and certificates from users in a secure fashion. There is no standard for how a DNS administrator communicates with end users about data for the zone. It is likely to be common that there will be a web page for submission of keys and certificates. This web page needs to be associated with some process (possibly automatic, possibly manual) that verifies that the submitted key or certificate is really associated with the person at the identified email address. In the commercial CA industry, this is often done with an automated mail message containing a single-use URL.

The DNS administrator also needs to convert those keys and certificates into SMIMEA records. Converting keys and certificates to an SMIMEA record is fairly straightforward, following the description in Section 2 of RFC 6698 and the steps in Section 3 of draft-ietf-dane-smime-07.

After the records are created or updated, they need to put those records in the DNS zone and resign the zone. The steps to do this are completely dependent on the DNS server software being used, the DNSSEC signing software being used, the DNSSEC signing policies, and so on.

Creating Certificates

The extension has a feature in its UI which allows the user to create a self-issued certificate that can be used for both signing and encrypting. When the user instructs the extension to create the certificate, the extension does the following:

1. Prompts the user for the e-mail address for which this certificate will apply.
2. Gets a random number with at least 128 bits of entropy from the cryptographic library.
3. Uses that number as a seed for generating a new 2048-bit RSA key pair.
4. Marshals the e-mail address and the public key into the "to be signed" part of a certificate. The contents of the certificate indicate that it is self-issued by filling both the subject and issuer names with e-mail address, and using the new public key for both the subject and issuer keys.
5. Signs the certificate contents with the new public key.

There is no standard mechanism for the user to transmit this new certificate to their DNS administrator, so this will have to be specified on a site-to-site basis. The site administrator needs to be be able to accept multiple certificates per e-mail address, needs to be able to convert certificates into SMIMEA records, and needs to be able to remove SMIMEA records on request.

Libraries Used by the Extension

The cryptography used in the extension will need to do many types of cryptography for encrypting and signing. The extension can use either an external cryptographic library such as OpenSSL (the most common external library) or a cryptographic library that is already part of the programming language. The cryptographic library needs to have a good source of entropy, or have access to the operating system's source of entropy, for creating the key pairs used when creating certificates.

The extension needs to be able to use read S/MIME messages. This requires a library that knows how to handle many of the oddities of MIME. Similarly, although more simply, the extension needs a library that can form MIME messages when creating signed or encrypted messages. The same library or libraries needs to be able to read and create the S/MIME-specific parts of a message, such as the combining of multiple keys for an encrypted message.

Additionally, the extension needs a library to validate DNSSEC on the SMIMEA records. There are a few such libraries, such as libunbound and the more modern getdns. The library has to be able to extract the certificate or key from the SMIMEA record; none of the libraries do so currently, but this is fairly easy to add.

Implementing Signing and Encrypting in a Mail User Agent

Most mail user agents provide ways to add new features for the user. The ways have a variety of names such as extensions, plug-ins, and apps, but they all share the same general structure.

Mozilla Thunderbird is a popular open source mail user agent. It is available on a wide variety of platforms, including Windows, MacOS, and Linux, and has a fairly well documented extension system, so we use it as our example here. The internal structure of Thunderbird is similar to that of their Firefox web browser, so much of the code for a Thunderbird extension could be adapted to a Firefox webmail extension. See the references section for links to documentation on creating Thunderbird extensions.

Application and Extension Structure

The Thunderbird user interface consists of two parts:

- the menus, toolbars, and buttons that control the application, known as the "chrome"
- the windows that show lists of folders or messages, message contents, and so forth

The chrome (no relation to the Google browser) is defined in XML User Interface Language (XUL), a specialized language invented by Mozilla. Thunderbird itself includes XUL descriptions of the standard interface elements, and each extension includes XUL descriptions of elements specific to the extension that are merged into the standard interface. Modifying the chrome involves writing new XUL and specifying where in the existing interface the XUL is to be inserted.

The windows are implemented essentially as web browser windows, each displaying content coded in the web's HTML format. The content of the windows are created by scripts in the Javascript language. In some cases, the HTML is written by the Javascript, while in other cases it copies HTML from messages or other sources. Modifying the windows can be done by adding or changing the HTML, or adding or changing the Javascript that generates HTML.

When an extension is installed, it associates Javascript functions with new chrome elements such as clicking a button, and can also associate them with existing operations such as opening a message. An extension is packaged as an "xpi" file, which is actually a zip archive that contains files of the extension's XUL and Javascript, as well as a manifest file that that says how the extension's files are to be installed, and a resource description file (RDF) that checks for compatibility with specific versions of Thunderbird.

Javascript is usually sufficient to implement an extension's features, but in some cases the limitations of Javascript require part of an extension to run directly on the host machine, typically coded in the C++ programming language. For example, an extension that needs to make direct calls to the DNS (such as to get an SMIMEA record) cannot do so in Javascript, and must use its own DNS code or call out to the operating system.

Mozilla provides a component object model called XPCOM that enables programmers to write modules in a variety of programming languages and call them from Thunderbird and other Mozilla applications. Despite the power and flexibility of native code, extension programmers avoid it whenever possible. One reason is that that the programming and debugging is considerably trickier than for Javascript. Another is that unlike Javascript, native code is not portable across operating systems, so separate versions have to

be developed for Windows, MacOS, and Unix/Linux systems. Most importantly, because native code provides access to the full underlying system, it presents security risks that Javascript, which runs within the Thunderbird environment, does not. However, when such code is needed (such as to make DNS queries), the extension library allows it.

Implementing Secure Mail as an Extension

The secure mail extension contains three main sections: incoming mail, outgoing mail, and key management.

Incoming Mail

For incoming mail, the extension examines each incoming message as it is fetched, and as it is about to be displayed. When a message is fetched, the extension checks to see if it is signed or encrypted, so that it can add appropriate annotations to the message index window.

When the message is about to be displayed, the extension has to prepare the message for display.

- If the message is signed, the extension removes the signature before passing the message on to the display code.
- If the message is encrypted, it creates a decrypted version of the message to pass to the display code.

In each case, indicators in the chrome show that the message was signed or encrypted, and whether a signature was valid. Note that some incoming messages are both signed and encrypted, so the chrome must have separate indicators for each status.

Outgoing Mail

For outgoing mail, the chrome contains controls to indicate that a message is to be signed or encrypted. When the user clicks the Send button, extension code encrypts, signs, or encrypts-and-signs the mail before passing it to the routine that does the actual mailing.

Certificate Management

For both incoming and outgoing mail, the extension has to maintain the user's own certificates, and to fetch the certificates for correspondents. Thunderbird provides ways for extensions to remember state information.

Certificate management adds a new chrome control and probably an associated window where a user creates a public/private key pair and a certificate containing the public key. The private key is stored locally, while the certificate is sent to the user's mail provider to be installed in the DNS. The most straightforward way to send the certificate is again to do it via web requests created by Javascript in the extension.

Javascript cannot directly do the SMIMEA DNS queries needed to fetch the keys. One possibility would be to write native code modules to do the DNS queries using XPCOM. Another approach, at least for prototyping, would be to use a remote web server as a proxy. Since Javascript can easily create URLs and fetch remote web pages, the Javascript would use a web based query for the required key or certificate.

The remote web server returns DNSSEC validation information along with the SMIMEA records, so the extension can check that the results are indeed valid.

Bibliography

IETF RFCs and drafts

Internet mail message format, RFC 5322. The format of all Internet mail messages.

MIME message format, RFC 2045 to 2049. The format of all MIME messages including S/MIME.

MIME Multipart/Secure, RFC 1847. The MIME container format needed for detached signatures.

S/MIME v3, RFC 5751. The format of S/MIME entities within a MIME message.

S/MIME certificates, RFC 5750. Defines the process for validating and using certificates in S/MIME.

Cryptographic Message Syntax (CMS), RFC 5652. The data format used for cryptographic signatures and encrypted data.

Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, RFC 5280. Defines the format of PKIX certificates.

DANE TLSA, RFC 6698. Defines the record types used to store certificates in the DNS.

DANE SMIMEA, draft-ietf-dane-smime. Defines the usage of DANE record types in S/MIME.

DNSSEC, RFC 4033 to 4035. Defines the DNSSEC security extensions to the DNS including most of the record types and the processing rules.

NIST Documents

Guidelines on Electronic Mail Security, SP 800-45.

Federal S/MIME V3 Client Profile, SP 800-49.

Secure Domain Name System (DNS) Deployment Guide, SP 800-81-2.

Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography, SP 800-56A.

Recommendation for Pair-Wise Key-Establishment Schemes Using Integer Factorization Cryptography, SP 800-56B.

A Profile for U. S. Federal Cryptographic Key Management Systems (CKMS), SP 800-152.

Recommendation for Cryptographic Key Generation, SP 800-133.

Mozilla Thunderbird Extension References

https://developer.mozilla.org/en-US/Add-ons/Thunderbird/Building_a_Thunderbird_extension_2:_extension_filesystem This page defines the structure of an XPI file and leads to many other references on how to created the contents of the XPI.

https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XPCOM This page gives an overview of XPCOM and points to other references on the use of specific programming languages for creating the extensions.