Smaug: Design Specification

Table of Contents

1 Executive Summary

Smaug is a library that implements bootstrapping secure key learning for S/MIME email through DANE. Smaug interfaces with Thunderbird (a Mail User Agent, MUA) through an extension, but is intended to be a general DANE S/MIME library. Our future plans are to integrate Smaug directly with Thunderbird's native S/MIME support. Because of this, Smaug's simple class structure is independent of any specific Thunderbird logic.

Smaug is a library (libsmaug), is written in C++, and includes bindings for C.

2 High-Level Design

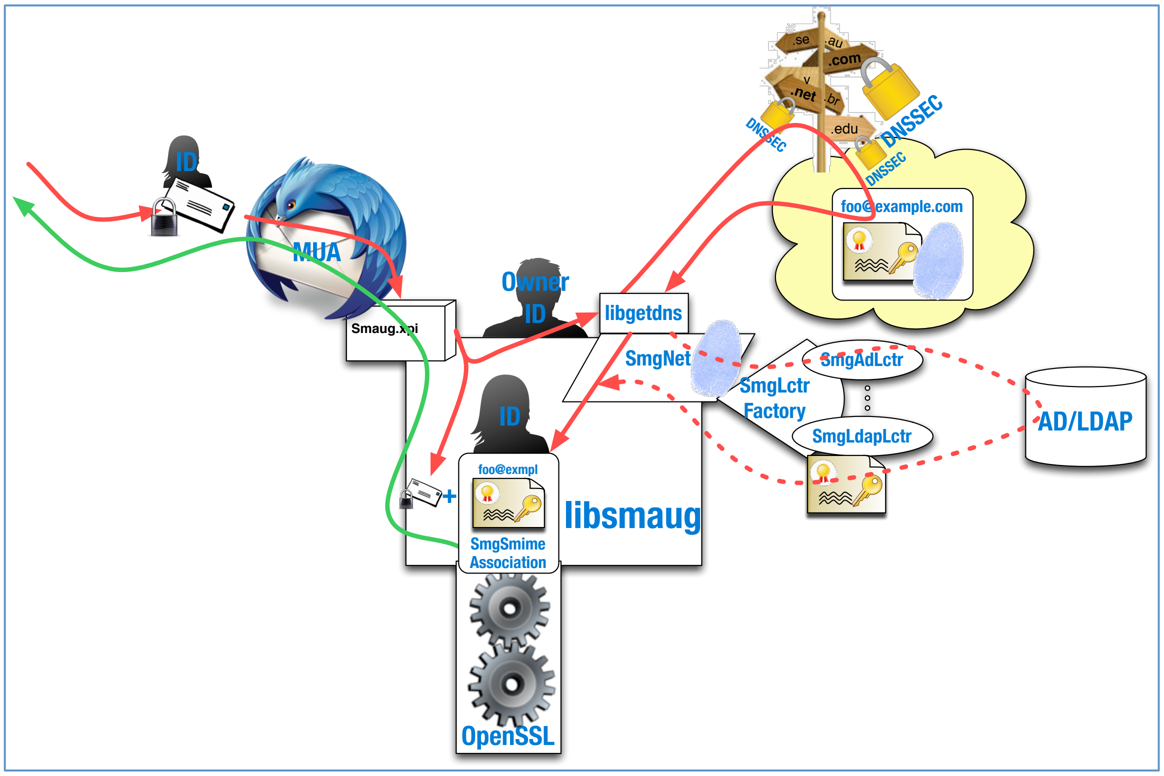

Figure 1 High-level design of Smaug and its internal components

The high-level control flow for Smaug is to find a DANE association for a given <email address, operation> (i.e. <alice@example.com, encrypt email>), then fetch the S/MIME certificate (if it is available and authorized by the DANE association), and then (optionally) perform a specified cryptographic operation (sign, validate, encrypt, or decrypt). The reason the cryptographic operation is optional is to facilitate integration with existing S/MIME codebases (which may already perform crypto after learning certificates).

Smaug's first operation is to convert a specified email address into a DNS domain name and query DNS for a corresponding SMIMEA record (the DANE record that describes S/MIME associations). This association will become a Smaug class instance (SmgSmimeAssociation), and it will contain association details (per the DANE specification) and either a full S/MIME certificate or a fingerprint of the certificate, and may also contain a locator that points to a network resource location that serves the certificate (i.e. Active Directory, LDAP, WebFinger, etc.). If the association specifies a locator instead of a full S/MIME cert, Smaug will attempt to fetch the cert (if the service protocol specified by the locator is understood). Finally, with the returned association object, the user can use Smaug to perform the full cryptographic operation.

3 Component Decomposition

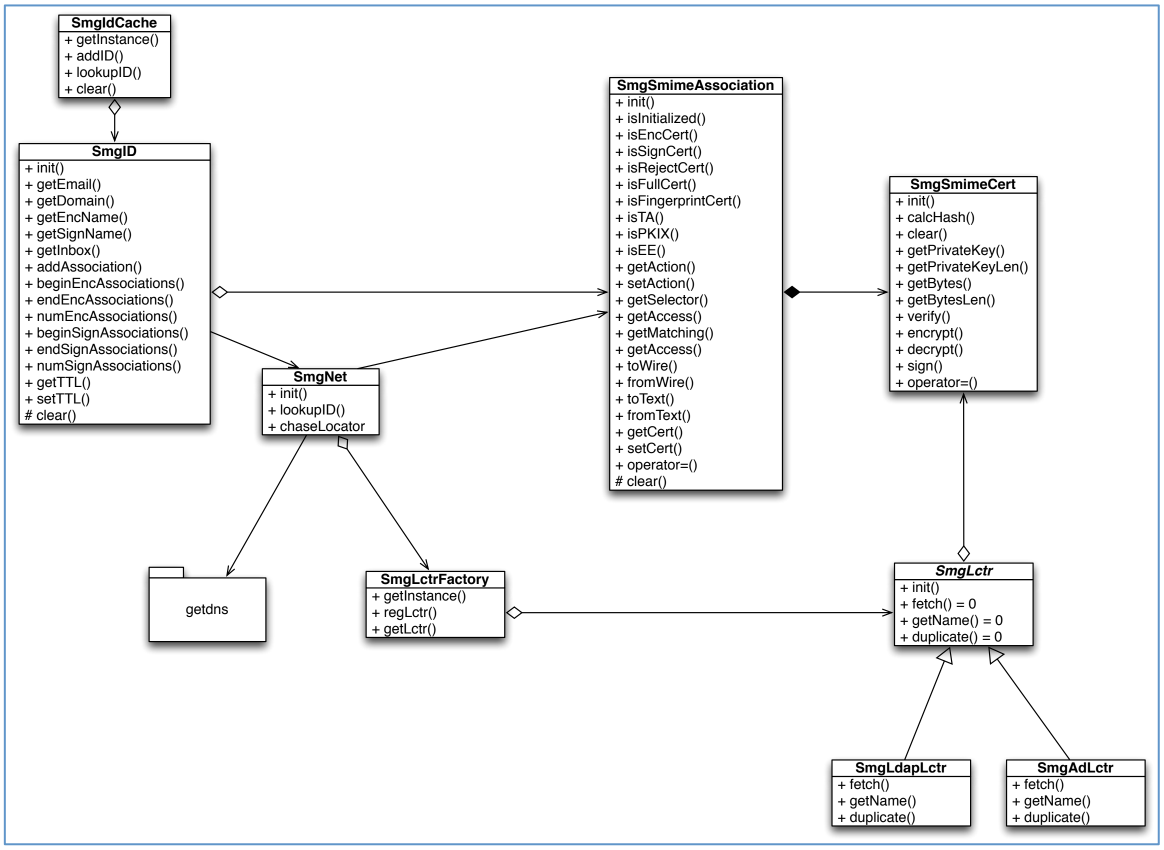

Figure 2 UML of classes of Smaug

3.1 SmgID

This class encapsulates the idea of an identity, for Smaug. Specifically, this is an email identity, and as a result it is uniquely identified by an email address. Instances of this class aggregate associations (which abstractly represent and aggregate keys).

- **init()**
  This method initializes the ID instance with an email address, and optionally pairs of local files for encryption/decryption/signing/verifying (if the ID is a local user).

- **getEncName()**
  This method returns the DANE formatted domain name for the encryption key discovery, for this ID.

- **getSignName()**
  This method returns the DANE formatted domain name for the signing key discovery, for this ID.

- **getInbox()**
  This method is a simple accessor for retrieving the ID's email address.

- **addAssociation()**
  This method adds a **SmgSmimeAssociation** object to either the internal encryption or internal signing association lists (depending on the values specified by the **SmgSmimeAssociation** itself).

- **beginEncAssociations()**
  This method returns an iterator to the beginning of the internal list of encryption **SmgSmimeAssociation** objects.

- **endEncAssociations()**
  This method returns an iterator to the end of the internal list of encryption **SmgSmimeAssociation** objects.

- **numEncAssociations()**
  This method returns the number of internal encryption **SmgSmimeAssociation** objects.

- **beginSignAssociations()**
  This method returns an iterator to the beginning of the internal list of signing **SmgSmimeAssociation** objects.

- **endSignAssociations()**
  This method returns an iterator to the end of the internal list of signing **SmgSmimeAssociation** objects.

- **numSignAssociations()**
  This method returns the number of internal signing **SmgSmimeAssociation** objects.

- **clear()**
  This method resets member variables and empties internal lists.

3.2 SmgIdCache

This class is a singleton class that caches recent IDs (respecting TTLs of associations).

- **getInstance()**
  This method is an accessor for the singleton's global instance.

- **addID()**
  This method takes a **SmgID** instance and adds it to an internal map (cached only as long as the specified TTL). If the TTL is 0, the ID is cached indefinitely.

- **lookupSmimeID()**
  This method looks up a **SmgID** instance by an email address. If no value is found, a NULL is returned. If a value is found, but its TTL has expired, the instance is removed and a NULL is returned.

- **clear()**
  This method clears and frees all internal state.

3.3 SmgNet

This class acts as the network abstraction layer for Smaug.

- **init()**
  This method initializes internal state.

- **lookupSmimeID()**
  This method takes a **SmgID** instance and an enumerated type (which describes the proposed operation, encryption/signing). With these inputs, the method formats a domain name, queries DNS, and parses the response (if there is one) into *n* **SmgSmimeAssociation** instances (where *n* is >= 0), and adds them to the **SmgID** input variable.

  This method should chase the Certificate Access field if it includes a URI.

- **chaseLocator()**
  This method takes an instance of **SmgSmimeAssociation** and uses its locator information (if there is any) to create a derived instance of the abstract class **SmgLctr** that corresponds to the service pointed to (LDAP, AD, etc.), from the factory class **SmgLctrFactory** (a singleton object). If a certificate is successfully returned, it is added to the **SmgSmimeAssociation** instance.

3.4 SmgSmimeAssociation

This class is an Abstract Data Type (ADT) that represents a DANE association between email IDs and S/MIME certificates. Specifically, this class is aggregated by **SmgID** objects, aggregates **SmgSmimeCert** objects, and represents the DANE semantics of the association between instances of these classes.

- **init()**
  This method initializes an instance. It specifies values retrieved from DNS, or a binary buffer retrieved from a DNS query (wire format), *or* if this is an instantiation of the "Owner ID" (the local entity that is running the library), the file paths to the owning certificate and private key information.

- **isInitialized()**
  This is a simple Boolean accessor that returns if this instance has been initialized.

- **isEncCert()**
  This method is a Boolean accessor that just tests if the association is an encryption association (specified by the domain name).

- **isSignCert()**
  This method is a Boolean accessor that just tests if the association is a signing association (specified by the domain name).

- **isRejectCert()**
  This method is a Boolean accessor that just tests if the association specifies that this certificate must be rejected.

- **isFullCert()**
  This method is a Boolean accessor that tests if the association contains a full certificate, as opposed to a fingerprint.

- **isFingerprintCert()**
  This method is a Boolean accessor that tests if the association contains fingerprint, as opposed to a full certificate.

- **isTA()**
  This method is a Boolean accessor that tests if the association references a Trust Anchor (TA). This could be in reference to either a PKIX or DANE TA.

- **isPKIX()**
  This method is a Boolean accessor that tests if the association requires PKIX validation.

- **isEE()**
  This method is a Boolean accessor that tests if the association references an End Entity (EE) certificate.

- **getAccess()**
  This method is an accessor that returns any "Locator" value that is specified in the "Access" field of the association.

- **toWire()**
  This method converts the association into wire-format, so it can be transmitted over DNS.

- **fromWire()**
  This method converts a binary buffer (containing an instance of the DNS record type SMIMEA) into this instance.

- **getCert()**

This method is an accessor the returns a pointer to the aggregated instance of **SmgSmimeCert**.

- **setCert()**
  This method is an accessor the sets an internal a pointer an instance of **SmgSmimeCert**.

- **operator=()**
  Used by the copy constructor.

- **clear()**
  Clears all internal state and memory.

3.5 SmgLctr (abstract class)

This is an abstract class that abstractly represents logic for following an SMIMEA's Access field. Specific protocols (like LDAP, AD, etc.) will be encapsulated in derived classes.

- **init()**
  This method initializes an inherited instance of this class with a URI reference.

- **fetch() = 0**
  This pure virtual method will be overloaded in derived classes to fetch an S/MIME certificate from external services.

- **getName() = 0**
  This method returns the "name" of this *class.* Derived classes will overload this pure virtual method so that they each have distinct names (used by the factory class). The value should be the protocol identifier that matches a URI.

- **duplicate() = 0**
  This method returns a newly allocated (on the heap) instance of this class. Derived classes will overload this pure virtual method so that they each return the appropriate classes when duplicated by the factory.

3.6 SmgAdLctr

This class is derived from **SmgLctr**, and is derived to interact with Active Directory forests.

- **fetch()**
  This virtual method will use the internal URI to communicate with an AD infrastructure, and retrieve user S/MIME certificates.

- **getName()**
  This virtual method returns a name that is specific to this class.

- **duplicate()**
  This virtual method will return an instance of this class when duplicated by the factory.

3.7 SmgLdapLctr

This class is derived from **SmgLctr**, and is derived to interact with LDAP directories.

- **fetch()**
  This virtual method will use the internal URI to communicate with an LDAP infrastructure, and retrieve user S/MIME certificates.

- **getName()**
  This virtual method returns a name that is specific to this class.

- **duplicate()**
  This virtual method will return an instance of this class when duplicated by the factory.

3.8 SmgLctrFactory

This class is a factory for dynamically getting a derived instance of the **SmgLctr** abstract class that is needed for a specified URI.

- **getInstance()**
  This method is a static accessor that returns a reference to the singleton's global instance.

- **regLctr()**
  This method adds a new instance of a **SmgLctr** object with a name that is taken from the calling the instance's **getName()** method.

- **getLctr()**
  This method takes a name as an argument and returns a duplicated instance of the corresponding **SmgLctr**.

3.9 SmgSmimeCert

This class is an ADT that encapsulates a public S/MIME certificate, and where available, also encapsulates the corresponding private key. This class also performs cryptographic functions using the encapsulated key material.

- **init()**
  This method initializes the certificate object with a byte buffer (if available), or with a file path(s) to the certificate file (and optionally a file path to the associated private key). If file either or both paths are specified, the respective data will be loaded by this method.

- **calcHash()**

This method takes and enumerated types of selector and matching parameters, and an output string. These inputs are used to calculate a specific type of hash (SHA256, SHA512, etc.) over this object's internal public key. If the object is not initialized with a full key, then this method fails. If there is a proper key, then this method returns the corresponding hash (as a string).

- **clear()**
  This method clears all of the internal state and frees dynamically allocated memory.

- **getPrivateKey()**
  If this instance has a private key loaded, this method will return a pointer to its bytes.

- **getPrivateKeyLen()**
  This method returns the length of the internal private key.

- **getBytes()**
  This method is a simple accessor that returns a pointer to the internal byte buffer of the public certificate.

- **getBytesLen()**
  This method returns the length of the public certificate information.

- **verify()**
  This method takes a byte buffer and signature buffer as input and uses the internal certificate to verify the signature.

- **encrypt()**
  This method takes a byte buffer and an output buffer as input and uses the internal certificate to encrypt the contents into an output variable.

- **decrypt()**
  This method requires that the private key be set. It then takes a byte buffer and an output buffer as input and uses the internal private key to decrypt.

- **sign()**
  This method requires that the private key be set. It then takes a byte buffer and an output buffer as input and uses the internal private key to sign.

- **operator=()**
  Used by the copy constructor.