OPEN SOURCE APPLICATIONS FOUNDATION

# Automatic Secure Email

Andy Hertzfeld 11/4/02

Even though secure email technology has existed for decades, and many email clients currently support it, the vast majority of email messages sent today still travel through the network in plain text, naked to the scrutiny of observers on the networks they pass through. What are the reasons for this? How can we improve the situation?

One reason is that security features are often complex to configure and administrate, especially for non-technical users who are unfamiliar with the underlying concepts. Many users don't value security highly because they don't understand the nature of their vulnerabilities. Even if you are technically inclined and concerned with security, you can't adopt secure email all on your own, since your correspondents must, too, if you want to continue to communicate with them.

So widespread use of secure email won't occur until it is adopted by a critical mass of mainstream users, and most mainstream users won't start using it until it is just as convenient and easy to use as the alternative. Widespread use of secure email won't really happen until commonly used mail clients deploy it automatically, by default, with minimal configuration required. That certainly seems like a worthy goal for Chandler.

The problem is that you can only send an encrypted message to someone who has set up secure email themselves and somehow made you aware of their public key, as well as knowledge of the particular formats they accept. Cryptographic key pairs are too complex for ordinary users to manage manually. There are also complications involving trusting that a given key is authentic. And even if you know a correspondent's public key, they might be receiving your message with an alternative client (perhaps a hand-held device) that uses a different key or doesn't support secure email.

In order to be able to automatically send messages securely, we need a way for our mail client to automatically determine the relevant public security information for each recipient. One possibility is for each client to register their public security information with a server, which could be queried by other clients before sending. However, that information could get out of sync with the client, and we prefer a solution that doesn't require additional servers if possible.

A better approach is to define a simple mechanism for email clients to communicate with each other directly, via email itself, so they can share descriptions of their capabilities, including the cryptographic information necessary to talk to them in a secure fashion. With such a mechanism, your email client could negotiate with your correspondent's client behind the scenes, to determine the best way to send a message securely.

Let's examine what such a mechanism would look like, without trying to specify it formally here. We can define a new, RFC822-style email header called 'X-Profile-Request', which is used to request a profile from the receiving email client. When an email client receives a message that contains a profile request header, it responds with a 'profile response', which is an email message containing an 'X-Profile-Response' header and the responding client's profile contained in xml file attached to the response. The profile includes the types of encryption algorithms and formats supported by the client with their corresponding public keys, as well as a user-agent string, a list of supported mime-types, and possibly other descriptive information like contact info expressed as a vCard.

Clients must guard against revealing too much information to arbitrary requesters, since you don't necessarily want to even reveal your existence to spammers or unknown parties. Profile requests contain a request identifier, generated by the requester, which is included in the profile response to help verify its authenticity; this helps guard against trouble-makers providing false keys in phony responses. In order to help protect ourselves against evil-doers sending out large numbers of profile requests, we also require a profile request to include a hash-cash string, which is a string containing numbers that are easy to verify but are relatively hard to compute. Still, some users will wish to approve all profile requests explicitly, so there should be a user interface for dealing with them, perhaps integrated with instant messaging presence subscription requests.

An email client will remember the results from prior profile requests, so it will usually only have to send one the first time you communicate with someone new, or when you explicitly decide to resynchronize (It can also issue them behind the scenes when you add a new contact). When the user sends a new message, the client will look up profile information for the recipients. If present, the profile information allows the client to send the message securely; if it's not present, the client will include a profile request in the message, so it can send it securely the next time. Alternatively, at the user's discretion, it could just send a profile request with an empty message the first time, and wait for the profile response so it doesn't have to send the message insecurely the first time.

Another complication arises from users having more than one email client for the same email address. We either need a mechanism for clients to synchronize their private keys (which might not be advisable, since it's hard to do securely), or to allow each client to have its own key pair and support multiple profiles associated with a given email address. In that case, when a client issues a profile request, it might get multiple responses, one from each recipient client that received the request. Then, when sending a message, the client must include multiple versions of the encrypted session key, one for each profile on hand for the designated recipient. There are some nice benefits for supporting multiple profiles per address that go beyond security, as it could be used to avoid sending large attachments to hand-held clients, for example.

In order to turn on encryption by default, we must be able to deal gracefully with the situation when someone receives a profile request or an encrypted message with a client that doesn't know how to handle it. There should be a simple way for recipients to notify senders that they received an encrypted message that they can't decode, and to request an unencrypted version. Encrypted messages could contain a brief, plain text message in the body area to explain what to do in the case. One possibility is for the recipient to reply with an empty message, and have our client recognize that and treat it specially, offering to send an unencrypted version.
We are interested in your feedback about the approach outlined above. Is automatic encryption as described above a worthwhile goal, or will it cause more trouble than its worth, especially during the early stages of adoption. What can be done to ease the transition?

Other open issues include determining the structure of the profile and what information it contains. Hopefully, we can leverage xml to allow an extensible framework, where clients can include optional information in custom namespaces. Is requiring a hash-cash puzzle in the request worthwhile or is it adding too much complexity? What kind of algorithm should it use?

We're also interested in an analysis from a security point of view. What kind of attacks is our scheme subject to, and what can we do to remedy them? Is there anything we can do to make "man in the middle" attacks harder?