Privacy Protected Email

Phillip Hallam-Baker, Comodo Group Inc.

Abstract

This paper describes an approach supported by open source tools that makes end-to-end email confidentiality practical. The solution described requires no additional user effort after the initial configuration and is compatible with the majority of deployed mail user agents. While email is not the only messaging application that requires security and confidentiality is not the only security concern, the approach described is readily extended to meet these requirements.

The solution described is designed to require absolutely no additional user effort when sending or reading messages and is both backwards compatible to the vast majority of installed email clients and forwards compatible enabling the deployment of new messaging protocols through the introduction of a policy mechanism.

Why don't users encrypt?

Although it is widely agreed that users do not use encrypted email because it is 'too hard', the reasons for that difficulty are rarely given with precision.

The cost of obtaining certificates from a Certificate Authority is often given as the reason for the lack of success of S/MIME [1]. Yet every S/MIME client permits the use of self-signed certificates and Comodo CA has provided free S/MIME certificates for many years with only limited takeup from users.

One major reason for the lack of use is that users are unaware that the option of sending email encrypted even exists, let alone that every mail user agent in widespread use has supported this capability for over a decade.

Another related problem is the network effect. Every email user can accept unencrypted email but the number of email users who can accept encrypted email is vanishingly small. Furthermore the email encryption protocol that is most widely used (OpenPGP [2]) is not the protocol that has achieved ubiquitous deployment (S/MIME).

The real problem is that the user who attempts to use encrypted email today is required to do all the work themselves and there is too much effort required. The user must first discover that their client has encryption capability or select and install the plug-in necessary to use OpenPGP. The user must then generate a keypair, manage distribution of the private component to each of their mail devices. In the case of S/MIME the performance must be repeated every year when the certificate expires.

Sending encrypted email is not much easier, every existing tool puts the onus of discovering if a recipient can accept encrypted email on the sender.

Finally, there is a pervasive attitude of defeatism in the standards community. Having failed to achieve an end-to-end email solution after four tries, many are not only prepared to give up, they are going to try to convince everyone else to give up as well.

End-to-end security is not a solution to every confidentiality attack but it is the only approach that can provide an email sender with an assurance that their message will be delivered confidentially or not at all. It is thus the only approach that is capable of meeting the confidentiality requirements that keep doctors, lawyers and other professionals clinging to their facsimile machines in the misguided belief that they will protect their client's data.

Usable Encryption without Infrastructure

One of the reasons that S/MIME has failed is that it made infrastructure a precondition for deployment. But infrastructure is built to meet the needs of existing users. The Web succeeded in part because anyone with access to machine connected to the Internet could set up a server to publish their material. One consequence of that requirement is that the early Web did not support search capabilities as rival network hypertext systems did.

Key Pair Generation, Distribution and Publication

The key management tool enables a user to receive encrypted mail. The planned features of this tool are:

- Import email configuration parameters from an existing installed client [*].
- Generate a personal public key hierarchy and emergency recovery codes.
- Tell the user their *strong email* address.
- Publish the public key portion to a specified support service [*].
- Import decryption keys and email configuration parameters into another device [*]
- Configure the email client to send and receive encrypted mail [*].
- Allow the user to specify an email security policy.

The features marked [*] are planned features that are supported in the architecture but not supported by the current prototype. As a result the current prototype does not yet meet the 'zero click' usability criteria but obviously could in time.

Strong Email Addresses

To send encrypted mail, the user must know the strong email address [3] of the recipient. This consists of a key fingerprint joined to a traditional email address using a query mark as a separator[1][2]:

```
AB7LRE-3EKR7K-ECT2KV2-7ATCFH-DXB?alice@example.com
```

[1] Use of PGP fingerprints is grandfathered for legacy compatibility
[2] Fingerprints are generated using SHA2-512 and truncated to 128 bits. The birthday attack is not relevant to this particular security application.

The query mark (?) is chosen as the separator as it is a legal RFC2822 local-part but effectively invalid in existing email use. It is thus possible to publish a strong email address through many (but not all) of the traditional channels for publishing email contact information and in all the email contacts directories tested.[3]

Like the https URI method, a strong email address has an implicit security policy that encryption is required. More precisely, use of the address requires that the message be sent under a security policy signed under a public key that matches the specified fingerprint. A security policy may include statements such as:

- Always send email encrypted under a key with fingerprint X
- Send email encrypted under a key with finger print X if condition Y is true
- The PGP format is supported under key X
- The S/MIME format is supported under key X with algorithm Z
- Only apply this policy if policy P does not apply
- The JSON-MTP protocol is preferred over SMTP
- The mail should be delivered to an alternative mailbox.[4]

The policy language allows a user to precisely control their security settings. For example a user who uses content filtering as part of their mail abuse control strategy might require all mail to be encrypted to the public key of the content filtering device except in specific cases where end-to-end encryption is to be used.

While the current prototype is limited to exchange of mail over SMTP, the policy language is designed to support a wide range of applications over multiple transport protocols.

Key and Policy Discovery

The key fingerprint only provides the means of verifying the public key. To make use of the public key the fingerprint must be resolved to obtain the corresponding Privacy Hardening Block (PHB) [4]. Discovery of the PHB may be effected through direct discovery or through use of a PKI.

The direct discovery mechanism is based on the ni URI scheme [5]. The strong email address shown earlier has the following ni identifier[5]:

```
ni://example.com/phb;AB7LRE3EKR7KECT2KV27ATCFHDXB
```

This identifier may be resolved using the http well-known scheme as follows:

```
http://example.com/.well-known/ni/phb/AB7LRE3EKR7KECT2KV27ATCFHDXB
```

The requirement to publish the PHB is the only part of the process that requires infrastructure support. Fortunately there are many services that provide email, albeit not necessarily the user's preferred mail provider. This is the requirement that motivates support for the delivery of messages to an alternative address.

---

[3] The prototype recognizes the alternative convention =phb=

[4] This might be considered a security hole but it is the key fingerprint that is authoritative.

[5] The dummy algorithm 'PHB' used in place of the ni algorithm specifier as the privacy hardening block contains an algorithm identifier as a prefix.

Mail Proxy

Since no existing email clients support the use of strong email addresses a plug-in or proxy is required. Since an application plug-in would only support a single mail client, the prototype uses a proxy. This is coded in C to facilitate reuse of the code in writing plug-ins and providing native support.

In the ideal use scenario, the proxy is installed on the same machine as the mail client that is sending the email and outbound mail is redirected through the proxy via either the TCP/IP loopback interface or hooking the system call.

In an alternative implementation the user might use a cloud based proxy on the understanding that this will not provide them with end-to-end security.

Infrastructure Improves Scalability

The prototype avoids the need for infrastructure because it only supports a direct trust model in which the receiver is already known to the sender and they have a sufficiently trustworthy out of band channel to exchange a description of the public key. While this is a huge improvement on the current state of affairs it falls short of the goal of ubiquitous end-to-end confidentiality and integrity.

Adding a PKI makes it possible to support exchange of email with a receiver that the sender has never met and allow an opportunistic encryption approach to be taken in which email is always sent encrypted if the recipient has published an encryption key.

One of the major factors that has deterred use of secure email has been the inability of the rival OpenPGP and PKIX [6] camps to come to agreement on a PKI model. Rather than choose a single winner, the Key Management Tool and Mail Proxy both support Web Service interfaces [6] [7] to a PKI or PKIs of the user's choice.

The second stage of the PPE project will focus on developing one PKI option based on the author's (possibly erroneous) view of the requirements [8]. One of the reasons that the OpenPGP and PKIX models have both survived this far is that they serve distinct use cases that are not fully covered by the other. The peer trust model is not appropriate for enterprise use, private individuals who are sufficiently concerned to use email encryption are not willing to put their faith in a trusted third party.

This scheme extends the PKIX model used in S/MIME to encompass the peer-trust model of OpenPGP. The result is then further reinforced by applying the transparency principle advanced in Certificate Transparency.

Further Work

This paper is a necessarily (and reluctantly) condensed version of work described in four published Internet drafts and numerous design notes. It is intended to be the starting point for reinventing Internet security, not the end result.

Other Protocols

The approach described in this paper is equally applicable to other application protocols including chat, voice, video and data level security. As the spread of SSL/TLS [9] to a diverse range of protocols beyond HTTP proves, once an infrastructure has been deployed to support one application protocol, extension to other protocols is straightforward.

Economic Benefits

By securing the Web, SSL/TLS has enabled much of what is now known as 'Internet commerce'. Email is an important Internet protocol but has failed to achieve its full potential due to the lack of confidentiality and authenticity provided. Utilities and other companies send out invoices by postal mail due to the lack of confidentiality provided. Deployment of an email security infrastructure to compliment email analogous to the security infrastructure that protects the commercial Web is likely to unlock significant economic benefits.

Bibliography

[1] B. Ramsdell and S. Turner, "RFC-5751 Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2," January 2010. [Online]. Available: http://tools.ietf.org/search/rfc5751.

[2] J. Callas, L. Donnerhacke, H. Finney, D. Shaw and R. Thayer, "RFC4880: OpenPGP Message Format," November 2007. [Online]. Available: http://tools.ietf.org/search/rfc4880.

[3] P. Hallam-Baker, "PRISM_Proof Email Key Generation and Publication," October 2013. [Online]. Available: http://tools.ietf.org/html/draft-hallambaker-prismproof-key-00.

[4] P. Hallam-Baker, "Privacy Hardening Block (Work in Progress)," [Online].

[5] S. Farell and e. al., "RFC 6920: Naming Things with Hashes," April 2013. [Online]. Available: http://tools.ietf.org/search/rfc6920.

[6] D. Cooper and e. al., "Internet X.509 Public Key Infrastructure Certificate," IETF.

[7] P. Hallam-Baker, "OmniBroker Protocol," July 2013. [Online]. Available: http://tools.ietf.org/html/draft-hallambaker-omnibroker-06.

[8] P. Hallam-Baker, "JSON Service Connect (JCX) Protocol," July 2013. [Online].

[9] P. Hallam-Baker, "PRISM Proof Trust Model," October 2013. [Online]. Available: http://tools.ietf.org/html/draft-hallambaker-prismproof-trust-00.

[10] E. Rescorla and T. Dierks, "The Transport Layer Security (TLS) Protocol," IETF.