A Brief Comparison of Email Encryption Protocols

Raph Levien
raph@c2.org
http://www.c2.org/~raph/

Raph Levien is a graduate student in Computer Science at the University of California, Berkeley. He is also author of premail, a Unix-based email encryption utility supporting the PGP/MIME, MOSS, and S/MIME protocols.

This document briefly reviews and compares five major email encryption protocols under consideration: MOSS, MSP, PGP, PGP/MIME, and S/MIME. Each is capable of adequate security, but also suffers from the lack of good implementation, in the context of transparent email encryption.

I will try to address issues of underlying cryptographic soundness, ease of integration with email, implementation issues, support for multimedia and Web datatypes, and backwards compatibility.

An additional grave concern is key management. Contrary to some beliefs, key management is not a solved problem. All of the proposals contain some mechanism for key management, but none of them have been demonstrated to be scalable to an Internet-wide email system. My belief is that the problems with key management do not stem from the classic Web of trust/certification hierarchy split, but the nonexistence of a distributed database (with nice interfaces) for holding keys. The encryption protocols also stand in the way of such a database, with key formats that are either overly complex, inadequate, or both.

PGP

PGP (Pretty Good Privacy) is, of course, the de facto standard for email encryption on the Internet.

PGP's underlying cryptography is quite sound - RSA (up to 2048 bits with the most recent implementation), IDEA with a 128 bit key, and MD5. PGP is entirely in accordance with the recent recommendations on minimum keylength, and in fact does not include a mode of operation in violation of those recommendations. This makes PGP (and, by extension, PGP/MIME) unique among the encryption protocols.

PGP is packaged in a single application (i.e. a single binary) which performs encryption, decryption, signing, verification, and key management. It does not depend on the existence of great deal of infrastructure. These factors have, in my opinion, been decisive in PGP's popularity.

However, PGP is still not suitable for fully transparent email encryption. The reasons are complex, and I will only touch on them here.

The main missing feature is the lack of MIME integration. Thus, PGP is not suitable for multimedia types other than US-ASCII text. PGP does contain some support for 8-bit charsets, but at cross-purposes with MIME. Signature checking of non-US-ASCII data is simply not reliable. To give an idea of this problem, the most recent international version (2.6.3i) tries several different character set conversions when verifying signatures, to see if any of them will work.

However, since a large fraction of email _is_ US-ASCII text, this feature alone probably does not explain the lack of deployment. PGP contains a number of implementation flaws (including silly things like not locking files, so that concurrent invocations fail). In addition, key management has some problems. Mostly, key management is hard to learn, time consuming, and requires a great deal of manual intervention. The "Web of trust" is supposed to fix this, by providing transitive trust of key authenticity. However, in practice the Web of trust has not delivered. In my experience (with many dozen correspondents), I have only had one or two keys transitively trusted.

A standard PGP-signed message consists of a "-----BEGIN PGP SIGNED MESSAGE-----" line, the signed text (subject to some canonicalization rules), a "----BEGIN PGP SIGNATURE-----" line, a version line, the PGP signature itself, and an "-----END PGP SIGNATURE-----" line. All this is in the message body. The headers indicate that the message is a standard 7-bit, us-ascii, text/plain message. Thus, mailers have to parse the message contents to identify the message as PGP signed, or to extract the signed parts. This is at cross-purposes with MIME. Mailer implementors are reluctant to include such ad-hoc extensions. Existing extensibility mechanisms, based on MIME, cannot be used.

PGP encrypted messages are similar - the identification as an encrypted message can only be made by parsing the message body.

One technical problem with PGP is its inability to support single-pass processing for some operations, because the data format includes a size field.

PGP/MIME

PGP/MIME is an effort to integrate MIME and PGP. There is a workable draft based on the MIME security multiparts, but the PGP/MIME mailing list is divided. Some particpants are happy with the existing draft, while others feel that other points in the design space (for the most part, labelling existing PGP message formats with appropriate MIME types) would be better.

The design space is large and complex, with many constraints on efficiency, simplicity, backwards compatibility, and functionality. It is not clear that a consensus will develop at all.

There are two implementations of the PGP/MIME draft: premail, and the PGPMIME reference implementation by Michael Elkins.

PGP 3.0

Many people are hoping that PGP 3.0 will somehow come along and solve all their problems. PGP 3.0 is only an evolutionary improvement over the existing implementations (MIT PGP 2.6.2 and PGP 2.6.3i). For the most part, it will support only the existing message formats. There may be support for decoding draft PGP/MIME signed messages, but this is still being negotiated.

The main advance in PGP 3.0 is a cleaned up implemenation. The PGP 2.6.2 code is disgusting, and should not be integrated directly into any mailer application. The 3.0 code will be modular and based on published interfaces. Furthermore, the 3.0 development team plans to release the code as both a stand-alone application and as a library.

It is difficult to predict when the official release will happen. Based on what I've seen, fall of 1996 seems the most likely.

MOSS

MOSS (MIME Object Security Services) is an attempt at an email encryption protocol in accordance with MIME. It is currently an Internet RFC. There is a reference implementation (TIS/MOSS 7.1).

MOSS is mostly cryptographically sound. However, the choice of symmetric encryption algorithm (and key size) is left unspecified. Thus, it cannot be said that MOSS is in accordance with the recommendations for minimum keysize. In fact, the only public implementation, TIS/MOSS 7.1, uses 56-bit DES, which is in direct violation of these standards.

The TIS/MOSS implementation has a number of other problems. It is big and complex, probably due to its TIS/PEM ancestry.

MOSS supports two modes of key management: X.509, and completely manual key management. In this way, it is a dramatic advance over PEM, which only supported X.509, but life for implementors remains hard. One feature which I believe is sorely lacking is a cryptographic hash of the public key as the basic unit of manual key management. Thus, people either have to trust the mechanism by which the key was delivered, or examine the base-64 representation of the entire key. I consider this to be a serious usability problem.

I have not seen any evidence of a production-quality MOSS implementation, either released or still in development. Others also question whether MOSS is "real," in this respect. I find this to be the shame, because I feel it is in many ways the simplest and most elegant approach to email encryption.

S/MIME

S/MIME is an attempt to graft MIME support onto underlying PKCS standards, which are in turn a backwards-compatible extension of PEM.

The only symmetric encryption algorithm mandated by S/MIME is 40-bit RC2. Thus, S/MIME is in violation of the key size recommendations. Further, RC2 has not been confirmed to be publicly known. If RC2 is not known, then an independent implementation of S/MIME is impossible. Fortunately, source code for an alleged implementation of RC2 has recently been posted to the Internet, resolving this problem, if it is authentic. If not, then my reservations remain.

The S/MIME FAQ claims as an advantage that any two S/MIME implementations are guaranteed to be able to talk to each other. This is true, and a point in S/MIME's favor (one not shared by MOSS and MSP), but then on the other hand in the worst case the NSA should be able to read that message for about a tenth of a cent.

S/MIME also recommends 56-bit DES CBC and 168 bit DES EDE3-CBC. This is good; any S/MIME implementation in accordance with the recommendations will conform to the keysize recommendations as well.

S/MIME remains firmly grounded in the X.509 certification hierarchy, although the FAQ claims that the guidelines for hierarchies are "more flexible" than in PEM, presumably through its use of X.509 version 3 certificates. The spec for X.509 v3 explicitly allows certification paths to be start in the local security domain of the public-key user system (just a fancy way of saying "Web of trust").

One cryptographic weakness of S/MIME is that eavesdroppers can distinguish between encrypted and signed-and-encrypted messages, and moree importantly, tell who the signers of the message were. This violates the principle of disclosing a minimum amount of information. PGP, PGP/MIME, and MOSS do not have this problem.

Probably the most controversial aspect of S/MIME is its signature format. An S/MIME signed message can either be an encoded PKCS #7 packet or a MIME multipart in which the first part is the data to be signed, and the second part is a complete PKCS #7 (section 10) signed message. This protects quite well against munging by mail transport, but has some problems. First, the existence of two signature formats requires either manual selection or some kind of configuration process. If the single-part PKCS option is chosen, then recipients without S/MIME capability will not be able to read the message. If, on the other hand, the multipart is chosen, then the size of the message is doubled. In addition, the fact that the two signed messages are identical is not enforced (if it were, mailer munging would cause too many signatures to fail). Thus, Eve can send Alice and Bob a message (M1, (M2, Signature(M2))). Alice, not having an S/MIME implementation, would see only M1. Bob, having an S/MIME implementation, would see only M2, for which the signature would check. Alice, being suspicious, might call Bob up on the telephone and ask whether the signature was really valid. Bob would of course say yes. Unless they compared notes on the contents, they would not notice the discrepancy. To my mind, this counts as protocol failure, and thus it is not possible to claim that S/MIME conforms to best cryptographic practice.

S/MIME gets everything almost right, and it's possible that it may be improved even further (of course, at the expense of having a somewhat unstable spec). Thus, it's a truly viable contender, especially given the massive support promised by Internet email implementors.

MSP

I don't know much about MSP, but have included this section anyway in the interest of completeness. MSP (Message Security Protocol) is an email encryption standard from the OSI world. The MSP specification itself (http://www.imc.org/workshop/sdn701.ps) seems quite firmly based on X.400 messaging protocols, but its proponents argue that it can be adapted easily to the Internet. There is a specification for a MIME wrapper, available as a Microsoft Word document (http://www.imc.org/workshop/SDN70413.DOC).

An additional concern with MSP is its complexity - it is arguably the most complex of the five proposals. Its proponents agree that MSP is complex, but argue that it is due to a richness of features.

At this point, I am unable to evaluate its underlying cryptographic soundness, partly because the specification of cryptographic algorithms is left open in the MSP document.

As is typical of OSI standards, the specification is quite abstract and difficult to read. In spite of all these problems, some vendors have committed to supporting this protocol. Thus, it would be premature to reject it out of hand.

Integration with mailers

Integration with mailers is quite difficult. In general, the mailer implementor will need to add specific features to support cryptography. Because of the restrictiveness of ITAR regulations, such an approach may not be practical for US developers, at least while supporting strong cryptography.

Perhaps the biggest feature required in the mailer is integration of key management and the "address book". If this feature is not implemented in the mailer, then two address books are required - one to select email addresses, and another to map email addresses to keys. I used this approach in premail, and found it to be the source of many usability problems.

Another feature that is required for fully transparent integration is caching of decrypted session keys. If not implemented, then the user interface delays in navigating a mail folder become unacceptable. To my knowledge, no implementation supports this feature.

One dimension in the design space is whether the cryptographic engine is tightly integrated with the mailer (i.e. shares an address space), or is a separate process that communicates with the mailer. Both approaches have been implemented. Both approaches are subject to numerous pitfalls, which have unfortunately not been entirely avoided.

These issues have more to do with implementation than with the encryption protocol, but I thought I'd mention them here, so that they are not actively thwarted.

The key database

Technically, the nature of the key database is orthogonal to the bits-on-the-wire specification of the protocol, but it is equally important to standardize in practice.

All of the protocols provide for both manual maintenance, and in most cases, some kind of automatic or semi-automatic assist. In my experience, it is not possible for this procedure to be fully automatic; at some point, the user has to make a decision about which keys are trusted and which keys are not.

My argument is that the protocol should facilitate making this process as easy and quick as possible. One application which has, in my opinion, got this exactly right is Netscape 2.0. When presented with an untrusted certificate, it pops up a dialog box containing all the human-readable information in the certificate, as well as a "certificate fingerprint," which is just the MD5 hash of the certificate, if I remember correctly. Then, you have a choice between allowing or not allowing connections to the site, and warning or not warning before sending data to the site. Go to the Options|Security menu, and click on "Site Certificates" to try this out yourself

Considering that the user has to maintain this trust information manually, a certificate fingerprint is the most user-friendly way to do it. I can call up the site on the telephone and ask them for the fingerprint. They could print it in their brochures, put it on business cards, or whatever. However, these channels are _not_ good for anything bigger than a few dozen hex digits.

I should point out that none of the protocols do a good job with this one. PGP has three ways of naming keys: a "fingerprint", the 8 least significant hex digits of the modulus, and the user id. The fingerprint contains a cryptographic weakness, meaning it is quite straightforward to generate collisions in all three namespaces. Even if the fingerprint were sound, then it still can't be used by PGP 2.6 as an index into the key database. As a result, the responsibility falls squarely on the user to keep the database free of name collisions.

This is one of the big usability problems when working with PGP. Most people don't understand the problems, and so their management of the key database is sloppy. To my knowledge, a bogus key attack has not been mounted against PGP, but if it were, I believe the confidence in PGP could be sorely shaken.

My recommendation is that every one of the email protocols supports a standardized cryptographic has of the public key, so that when the time comes to ask the question "Do you, the user, trust this key?", the user has some idea what key is being referred to. On the cypherpunks mailing list a few months ago, I gave the specific proposal of using the MD5 hash of the MOSS representation of the public key including the "PK," prefix, but with all whitespace eliminated. I feel that this is a concrete proposal, and easy enough to implement, both for MOSS and PGP.

S/MIME's use of X.509 certificates would seem to help with usability, but it's still not clear to me how well email addresseses will map to X.509 distinguished names. No doubt this will be answered when there is more experience with using the protocol.

Conclusion

All of the proposals described here can be used for secure email. None of them will be widely deployed in this capacity unless they are implemented well. I have concerns that both MOSS and S/MIME are susceptible to political pressure which will restrict key sizes insecurely in practice. I would like to see consensus develop around one of the proposals, so that energies used for implementation can be more focussed and effective. It is my hope that the Internet Mail Consortium Security Workshop will move in that direction.

[Home] [Previous]