ARKE: A Proposal for Simplified OpenPGP E-mail Security

Dan Berger
dberger@cs.ucr.edu

March 11, 2003

Abstract

Since it's release in 1991[11], PGP (Pretty Good Privacy) has seemingly held the promise of widespread email security. Despite this, the actual rate of adoption of PGP has been slow. We contend that there are two prime factors contributing to this poor adoption - that of perceived complexity, and inter-operable end-to-end integration. This paper proposes a system to facilitate wide-spread deployment of OpenPGP compliant email encryption across platforms and mail user agents. The system consists of an SMTP proxy; capable of seamlessly encrypting outgoing mail on a "best-effort" basis, and a corresponding POP3 proxy; capable of performing decryption and assisting in key management.

1 Introduction

Since it's release in 1991[11], PGP (Pretty Good Privacy) has seemingly held the promise of widespread email security. In fact, one of Phil Zimmermann's reasons for developing and releasing PGP was to encourage widespread use of email encryption:

> What if everyone believed that law-abiding citizens should use postcards for their mail? If some brave soul tried to assert his privacy by using an envelope for his mail, it would draw suspicion. Perhaps the authorities would open his mail to see what he's hiding. Fortunately, we don't live in that kind of world, because everyone protects most of their mail with envelopes. So no one draws suspicion by asserting their privacy with an envelope. There's safety in numbers. Analogously, it would be nice if everyone routinely used encryption for all their E-mail, innocent or not, so that no one drew suspicion by asserting their E-mail privacy with encryption. Think of it as a form of solidarity.[15]

Despite the free availability of the software tools, the rate of adoption of PGP among the Internet community at large has been slow. PGP Inc. reports that their products are in use by "thousands of enterprise and government organizations, and millions of individual users"[12], to give a sense of the scale of the potential audience; the CIA estimates that the US alone had 166 Million Internet users in 2001[3].

Of course, PGP is no longer simply an application - RFC2440[7] defines the OpenPGP message format, and at least two open source projects; OpenPGP[4] and GnuPG[8] report to implement this standard. Still, if we aggressively assume that each of these applications is similarly in use by "millions of individual users" - an assumption that's clearly unfounded as OpenPGP is still alpha quality - the overall percentage of Internet users taking advantage of strong encryption to protect their personal correspondence is insignificant.

We contend that there are two prime factors contributing to this poor adoption - that of perceived complexity, and a lack of inter-operable end-to-end integration. In this paper we propose a system which addresses these issues, offering a simplified means of aiding the adoption of OpenPGP for email security. It

is assumed that the reader is familiar, at least at a high level, with PGP[1] in particular, and cryptography in general.

The rest of this paper is organized as follows. Section 2 briefly examines the two problems identified above - namely complexity and inter-operable end-to-end integration - and describes existing solutions. Section 3 discusses the proposed solution at a conceptual level, and briefly describes its inspiration and evolution. Section 4 enumerates the design criteria and goals which drove development of the described system. Section 5 discusses the construction of the proxies, and section 6 discusses the installation/configuration process. Finally section 7 discusses lessons learned and future work before we conclude in section 8.

2 Explaining Slow Adoption

2.1 Of Complex Interfaces and Mailer Integration

After a particularly contentious origin, PGP version 2.6 - became available to US citizens, free for personal use, by way of MIT circa 1994. This initial version, consisting of only a command-line interface, proved difficult to use in practice.

The prevailing Unix philosophy of stringing together single-purpose command-line tools meant that integrating PGP with a mail user agent (MUA) initially required the end-user to write the integration glue; typically in the form of a shell script designed to imitate the command-line signature of message editor or mail transport agent (MTA).

Over time, projects appeared which centralized this effort - producing shell scripts which would provide some level of PGP integration with the popular console-based MUA's of the time. Despite this, these MUA's typically didn't include this integration script in their distributions or default configurations, thereby forcing a user to obtain, install, and configure an additional component - assuming they even realized what PGP was and how it could benefit them.

On platforms such as Windows and Macintosh System - known for monolithic applications lacking standard extensibility mechanisms, end-user integration of PGP with their chosen MUA was typically impossible. A dedicated user might encrypt the occasional sensitive message, but pervasive message signing and encryption simply presented too large an inconvenience.

In December of 1996, shortly after the release of RFC2015[5], version 4.5 of PGP was released for the Windows and Macintosh platforms. It included a simplified user interface and a plug-in to support message signing/verification and encryption/decryption in Eudora[11]. This approach, of integrating PGP with MUA's through MUA-specific plug-in interfaces, continued with the release of a PGP plug-in for Microsoft Outlook in 1996 by Network Associates, who had acquired PGP from the failing PGP Inc. In 2000, the official list of supported MUA's again grew with the release of a Lotus Notes plug-in.

Nearly 10 years after it's initial release, only three mail user agents were officially supported by PGP. In the intervening years, other mailers had taken on the task of integrating PGP themselves, many without the guidance provided in RFC2015.

2.2 End-To-End Interoperability

The cumbersome user interface, and lack of consistent integration method caused a more fundamental problem which stymied adoption. At the end of 1996 there were at least three common methods for sending PGP signed or encrypted messages.

[1] In this document we will use the terms PGP, GnuPG, OpenPGP interchangeably. Similarly, when discussing the structures described in RFCs 2015 and 2440, we will refer to them collectively as "PGP/MIME". If it becomes important to distinguish a particular implementation of OpenPGP, or of the associated MIME standards, that distinction will be made in the surrounding text.

1. PGP blocks placed directly in message bodies - so-called “in-line-PGP.”

2. `application/pgp`: A withdrawn MIME[10] Content-Type.

3. The methods described in RFC2015, also referred to as “PGP/MIME”.

As one might expect - these three methods were only marginally inter-operable. The majority of mailers which claimed to integrate with PGP did so with either method (1) or (2) - very few initially made the effort to implement the new PGP-MIME standard. The end result was that PGP signed or encrypted messages were typically difficult to handle for both sender and recipient. As such, only a small group of dedicated individuals, who felt they had a “need” for strong cryptography, persevered in using PGP.

To add confusion, in 1998 a set of informational RFCs were released detailing a scheme called `S/MIME`[9]. While these RFC’s were not open standards - they relied on encumbered algorithms and therefor couldn’t be implemented without appropriate licensing - their release demonstrated that more groups were thinking about enabling email encryption. It also quickly became clear that PGP had little chance of being inter-operable with their efforts.

3 Aiding Adoption

Before we describe the proposed solution, recall what PGP offers its users.

1. **Authenticity and Non-Repudiation:** by digitally signing an outgoing message, an individual allows the recipient to verify that the message actually originated at the claimed source, and was not modified in transit. Additionally, the recipient; being in possession of a signed message, can prove to a third party that the claimed sender did, in fact, send it.

2. **Privacy:** by encrypting an outgoing message, the sender insures that only the intended recipient can access the message contents.

It is important to note that email, being by nature a store-and-forward medium, does not lend itself to being secured by simple application of transport layer security at the end points. Specifically, while an SMTP or POP session between Alice and her local servers may be encrypted via SSL, in the case where Bob is not a local user it is reasonable to assume that the message will be transfered between servers over a clear channel. It is similarly reasonable to assume that if the message is stored at any traversed server, that storage is in clear-text and available for intentional or accidental disclosure. Even in the absence of persistent storage, it is possible for the administrator(s) of the traversed servers to inspect messages as they are processed. PGP and S/MIME, therefore, encrypt messages prior to transmission - with only control information left in clear text to facilitate correct routing. While leaving the control data in the clear permits interoperability with the existing mail transport infrastructure, a purist will note that it leaks a non-trivial amount of information to an eavesdropper.

In principal, it is possible to insert the encryption/decryption and signing/verification at any point along the transmission chain. Doing so in a way which does not significantly weaken the authenticity and privacy guarantees is the focus of this proposal.

To illustrate the notion at a high level, consider the following situation. The author of this paper desires to communicate with an associate (we’ll call him Jakob) via secure email. The author uses a MUA which implements PGP/MIME, but his associate does not. They have a finite set of options, listed in order of convenience:

1. The two can agree to eschew security, transferring data as unsigned plain-text.

2. Jakob can obtain, install, and configure a PGP/MIME integration add-on for his MUA, if available.

3. Jakob can choose an alternate MUA which does implement PGP/MIME. If the desire to have secure communication outweighs the instantaneous and ongoing inconvenience of switching MUA's, this may be viable, but is clearly less than optimal.

4. Jakob and Dan can agree to not use PGP/MIME, but instead to compose and encrypt/decrypt messages outside the context of their respective MUAs. Practically, this entails using a standard text editor to compose or view messages, and using PGP to encrypt or decrypt them prior to transmission or upon receipt, respectively.

We propose an additional option - offering similar functionality to a MUA plug-in without being MUA specific. Rather than attempting to integrate with each MUA, we propose to perform encryption and signing at the hand-off point from the MUA - by way of a local SMTP proxy which processes outgoing messages - encrypting and signing according to a simple set of rules configured by the user.[2]

Similarly, rather than integrating signature verification and decryption with each MUA, we propose to perform these operations at the last point possible before the message is handed to the MUA - by way of a local POP3 proxy which processes incoming messages; verifying signatures and decrypting messages as appropriate.

In the next section we describe the guiding design principals, as well as discussing some inherent limitations of this scheme.

4 Ideas into Practice

A short list of goals drove the design process:

1. Cross Platform

2. Minimal Required Configuration

3. Minimal User Interaction

4. Preserve, as much as possible, the security semantics of tight mailer integration.

4.1 Cross Platform

While the author primarily uses Linux, for which there exist several mailers boasting comprehensive PGP integration[13][6], the majority of the target audience does not. Therefore it was obvious that the tools used to implement the solution should be portable across at least Linux, MS Windows, and Mac OS X.

Since some (minimal) amount of user interface was anticipated, this meant that a GUI toolkit needed to be selected which was either already available on the target platforms, or could be installed easily.

4.2 Minimal Configuration

While flexibility through configuration is beneficial to "advanced" users, the overwhelming design goal is to keep things as simple as possible. Optimally, the proxies should be as close to zero configuration as possible.

[2] Others have attempted to add this functionality to the MTA, one such example is [14] which seems to have been abandoned as of August of 2001.

4.3 Minimal User Interaction

The goal here is to hide as much of the operation of the proxies as is safe from the end user. While some interaction - such as prompting for their pass-phrase - is unavoidable, those events should be minimized. Once the needed information has been gathered, the system should do it's work silently - only demanding input when absolutely necessary.

4.4 Security Semantics

Since encryption/decryption is being done outside the MUA - and without it's knowledge - it is impossible to precisely imitate the security semantics provided by that tight form of integration. For example, messages stored in a "Sent" folder will be neither signed nor encrypted. Similarly, incoming messages will be decrypted prior to hand-off to the MUA and will be stored in clear-text in the users mailbox.

Aside from these necessary deviations, it was desired that the proxy system behave as much like a well-integrated mailer as possible. Specifically, it should be impossible for a sender to convince the recipient that a message was signed when it was not. It should similarly be impossible for a sender to convince the recipient that a message was transmitted encrypted when it was not. Practically speaking, this means sanitizing incoming message headers used to communicate between the proxy and user via the MUA.

5 Implementation

5.1 Python: The Language of Choice

It was decided to write the proxies in python[2] - an interpreted scripting language. This decision was made for several reasons:

1. Cross platform: the Python interpreter is available and actively supported for all platforms of interest.
2. Built-in GUI toolkit: the Python interpreter ships with support for TKinter - a Python binding layer on top of the Tk GUI toolkit. While not nicely integrated with native look and feel on all target platforms, it was deemed sufficient for the limited user interface required.
3. Existing PGP integration: GPL code exists to access PGP functionality by way of the gnupg command-line tool[1]. This avoided the need to write this layer of glue code.
4. Rich class library: Python has a fairly rich class library for dealing with strings, regular expressions and (most importantly) MIME message parts. The ability to leverage this class library significantly trimmed development time.

5.2 Structure

The first task was to write a protocol agnostic line-oriented network proxy. This class (`LineOrientedProxy`) handles all the common behavior - listening on the local side, establishing a connection to the remote server, etc. - and uses a callback system - allowing the user to register handlers for protocol-specific commands.

Inheriting from this are two classes - `SmtpProxy` and `Pop3Proxy` which specialize the `LineOrientedProxy` class to handle the SMTP and POP3 protocols, respectively.

Layered on top of these are two more classes - `EncryptingSmtpProxy` and `DecryptingPop3Proxy`, which implement the integration with PGP to provide message signing and encryption, and message decryption and signature verification, respectively. These classes extend the callback idea to allow the user to provide methods to call to prompt the user for a pass-phrase when needed. Since minimal interaction was a

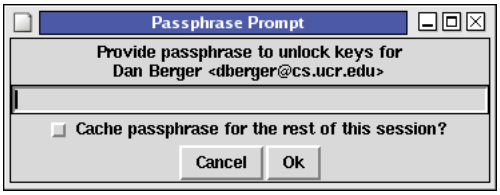

Figure 1: Passphrase Prompt Dialog

design goal - these classes also assume that the pass-phrase callback may remember the input it gathers. As such, these classes provide a means to ask that provider to forget credentials - for example if an incorrect pass-phrase was detected during signing or decryption.

Finally, two top-level classes `ArkeSmtpProxy` and `ArkePop3Proxy` provide the needed glue to string the components together into a working whole. These top level scripts also handle retrieving configuration file values and, in POSIX environments, releasing root privilege after binding to the needed sockets.

5.2.1 User Interface

The user interface portions of the code are isolated into three classes: `PassphrasePrompt`, `KeyringManager`, and `SendPrompt`. As the name suggests, `PassphrasePrompt` is responsible for prompting the user for their pass-phrase to unlock their secret key1. Credential caching can be enabled or disabled during instantiation.

`KeyringManager` is responsible for scanning incoming messages for PGP key blocks. Upon detection, the `KeyringManager` gives the user the opportunity to add the detected key to their keyring and use it for future signature verification and encryption2. In order to sign the imported key, the KeyringManager prompts the user for their pass-phrase using an instance of the `PassphrasePrompt` with caching disabled. This policy decision is currently hard-coded in the `KeyringManager`, but could become a configuration option.

Finally, the `SendPrompt` class is used to prompt the user to confirm if an outgoing message should be sent in clear-text if encryption keys for all recipients aren't available3. This preference can be remembered for the run of the proxy to avoid persistent prompting.

5.3 Communicating With the User

Aside from the explicit user interface described above, the POP3 proxy in particular needs to communicate the results of signature verification and decryption to the end user. Since we're operating outside the mailer through a standard protocol, there weren't too many options to explore.

The proxy adds SMTP headers to indicate the results of signature verification and to indicate if the received message was encrypted.[3] We assume that an interested user can configure their MUA to display these (or all) headers if desired.

As an additional bit of feedback, the proxy modifies the `From` header upon receipt of a signed message. It moves the original `From` header to `x-claimed-from` and populates the `From` header with the user id reported by gpg during decryption.

The proxy also inserts the string `''[Valid Signature]''`, or `''[Invalid Signature]''`, as appropriate, into the `From` header to indicate the state of the gpg signature.

Regardless of the message being signed - the `From` header is searched for the string `''[Valid Signature]''` and it is removed prior to message delivery.

[3] `x-pgp-signature-status`, and `x-pgp-encryption-status`, respectively.

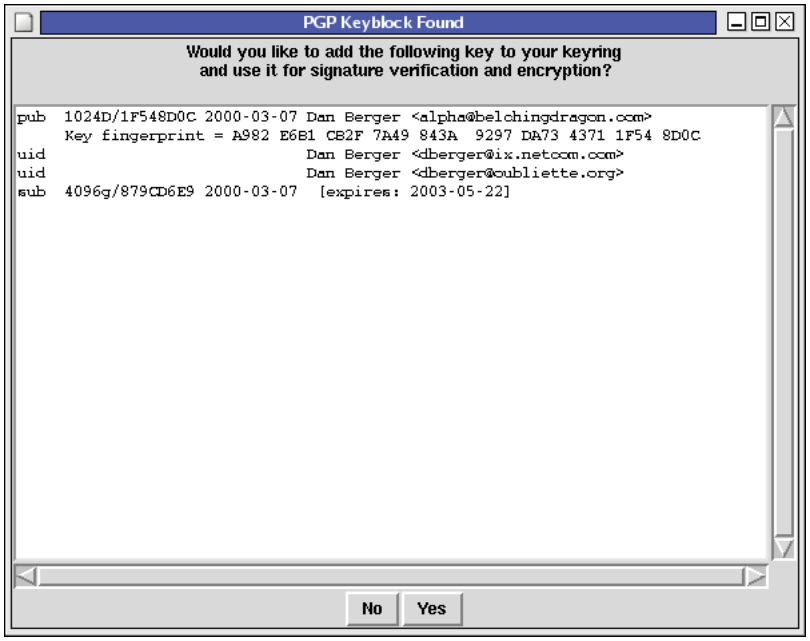

Figure 2: Key Import Dialog

Figure 3: Unencrypted Send Confirmation Dialog

6 User Installation

In their current form, the proxies are "installation heavy" - requiring more effort than is ultimately desirable. The procedure is:

1. Install Python 2.2.x as appropriate for the host OS. Installers exist for Windows and Mac OS X, binary packages are available for most flavors of Unix, and source is available failing all those.
2. Ensure GnuPG is installed and functional for the host OS. An installer is available for Mac OS X, a command-line binary is available for Win32, and binary packages are available for many flavors of Unix. Failing this, source is similarly available.
3. Generate a gnupg key-pair if needed.
4. Place the `.py` files for the proxies in the directory of your choice.
5. modify the `arke.cfg` configuration file - providing at least the gpg key id you wish to use and, under POSIX environments, the user and group that the proxy should become after privileged operations are complete. The proxies look for this configuration file in /.arke/arke.cfg in POSIX environments, or as specified by the `ARKECONFIG` environment variable.
6. Start the proxies by running `ArkeSmtpProxy.py` and `ArkePop3Proxy.py`

7 Lessons Learned, Work to be Done

The biggest fact reinforced during implementation is that encryption must be bitwise correct - almost isn't good enough. While testing interoperability with Evolution[13] and Mutt[6] much time was spent determining exactly what portions of the MIME structure was being passed off for signature - this required consulting the relevant RFCs, visually examining output data, and ultimately running mutt in the debugger with a few carefully placed breakpoints.

The code, in it's current form, is feature complete but should be characterized as pre-beta quality. There is a known issue handling large message downloads over POP - once a reproducible test case is found, this will be corrected. Additionally, the code is not as robust in the face of errors as it could (and should) be. It attempts to make up for this by being very conservative - always keeping the original message intact, and falling back on delivering it unmodified if needed.

One reasonable feature addition would be support for transport layer security. An informal survey of ISP's suggests that support for POP3 and SMTP over SSL is rare, hopefully this oversight will be rectified - at which point these proxies would be non-functional. Fortunately, adding SSL support should be straightforward, provided that there exists an SSL socket library for Python.

Additionally, as was mentioned briefly in section 5.2.1, some policy decisions are currently hard-coded (such as not allowing credential caching for the purposes of keyring management). These policies should be moved into the configuration file.

8 Conclusion

Since it's release in 1991[11], PGP (Pretty Good Privacy) has seemingly held the promise of widespread email security. Despite this, the actual rate of adoption of PGP has been slow. In this paper we propose that there are two prime factors contributing to this poor adoption - perceived complexity, and inter-operable

end-to-end integration. We propose a system to facilitate wide-spread deployment of OpenPGP compliant email encryption across platforms and mail user agents consisting of a set of SMTP and POP proxies which perform the required encryption and decryption work and assist in key management. We have built an initial version of this system and report on it's design and functionality.

References

[1] Gnupginterface. http://py-gnupg.sourceforge.net/.

[2] Python. http://www.python.org.

[3] U.S. Central Intelligence Agency. World Fact Book 2001. http://www.cia.gov/cia/publications/factbook/geos/us.html.

[4] Cryptix. OpenPGP. http://www.cryptix.org/products/openpgp/index.html.

[5] M. Elkins. RFC2015: MIME Security with Pretty Good Privacy. http://www.ietf.org/rfc/rfc2015.txt.

[6] Michael Elkins. mutt. http://www.mutt.org.

[7] J. Callas et. all. RFC2440: OpenPGP Message Format. http://www.ietf.org/rfc/rfc2440.txt.

[8] Free Software Foundation. Gnu Privacy Guard. http://www.gnupg.org/.

[9] S/MIME Working Group. http://www.imc.org/ietf-smime/index.html.

[10] IETF. RFC2045-2049: Multipurpose Internet Mail Extensions (MIME). http://www.nacs.uci.edu/indiv/ehood/MIME/MIME.html.

[11] PGP Inc. PGP History. http://www.pgp.com/display.php?pageID=49.

[12] PGP Inc. PGP Marketing Document. http://www.pgp.com/display.php?pageID=2.

[13] Ximian Inc. Evolution. http://www.ximian.com/evolution.

[14] tom@lemuria.org. Tea: Transparent encryption agent. http://www.neurosis.org/dvd/mirrors/lemuria.org/Software/Tea/, 08 2001.

[15] Philip Zimmermann. *PGP User's Guide Volume I: Essential Topics*, October 1994.