New Directions in GnuPG Development

Werner Koch
g10 Code
<wk@gnupg.org>

February 12, 2002

Abstract

The GNU Privacy Guard is nowadays a well known tool for encrypting and signing data. GnuPG is based on the OpenPGP protocol which is rightfully favored over S/MIME. However, S/MIME is getting more in use and so we have worked on an extension to support this protocol too. This paper discusses the new architecture we have chosen to accomplish this task; it is modular design which will closely integrate both protocols and allow for hardware tokens.

1 Introduction

All over the world Free Software is getting more and more attention even by governmental administrations. The German Federal Government was about one of the first to support a Free Software project and is now trying to promote the use of Free Software due to its better security, cost effectiveness and standard conformance. With the move of the capital from Bonn to Berlin a huge demand for secure electronic communication was caused. The BSI (Federal Agency for Information Security) started a pilot project for desktop to desktop security named *Sphinx*. This is based on a tailored version of the MailTrusT v2 X.509 profile and believed to be mostly inter operable with most other X.509 based systems.

A few *Sphinx* compliant proprietary implementations are now available for MS Windows. However, to migrate to GNU/Linux systems on the desktop an appropriate implementation is needed. The BSI requested a full GPLed system and contracted g10 Code[1], Intevation[2] and Klarälvdalens Datakonsult[3] to develop such a system. Due to the planned migration of at least one ministries IT infrastructure to GNU/Linux, the project has a very tight schedule so that the first public demonstration will be done by mid March at the CeBIT.

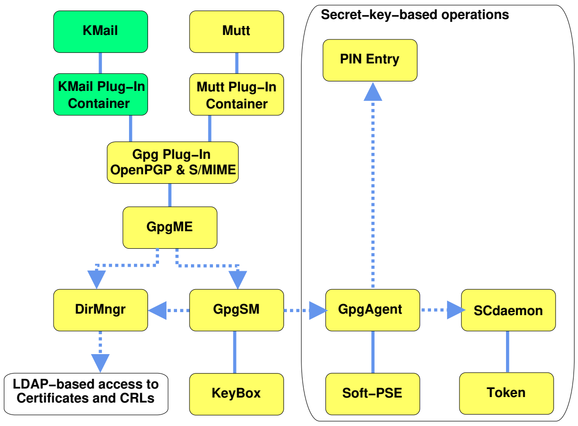
[1]http://www.g10code.com
[2]http://www.intevation.de
[3]http://www.klaralvdalens-datakonsult.se

2 Design Overview

It should not be assumed that a desktop box is only used by a single user; it is more likely that the same physical box is shared over time by several users. So we must take care that the confidential information of one user are hidden from other users. We can't lock out root, though. We also assume that local exploits are fixed in time and that remote exploits won't happen ;-).

By using smartcards we can safely protect private keys from any remote exploits. There is nothing we can do against remote exploits (except not to introduce too many bugs with our system) and rogue admins manipulating code or data. To some extend we can implement a detection of changed data. A better assurance against forged signatures is only possible with a trusted handheld used to view and sign a document. Protection of encryption is even more complicated because it is unlikely that a user will create or work with the plaintext version of a document only on a handheld.

Here is a diagram giving an overview of the new architecture:



A central point is GPGME which acts as middleware between the frontend applications and the crypto backend. On top of GPGME a plugin is used which provides most of the functionality required by the *Sphinx* specification. It is designed as dynamically loadable plugin to avoid packaging and upgrading headaches of different KDE versions.

GPGME itself delegates the crypto operations either to GPG when an OpenPGP[1] key is used or to GPGSM for S/MIME[5] certificates. GPGSM does the actual S/MIME handling with the help of GPGAGENT for private key operations and DIRMNGR for revocation lookups. Certificates are stored in a database.

Smartcard access is delegated to yet another program called the SCDAEMON. A PKCS-11[3] library will eventually be added to the system so that PKCS-11 aware applications will be able to make use of the system.

The dotted lines above mark module communication by employing the so called Assuan protocol; a simple text based protocol similar to FTP. The communication can be done by forking an instance of the server and using pipes or by starting the servers in advance and using Unix domain sockets for communication (which makes most sense for DirMngr, GpgAgent and SCdaemon). This gives us the nice property of having processes as security barriers and allows for easy migration of modules to external devices. A small support library makes the use of the protocol easy and allows us to add an extra security layer to the protocol without changing the actual program code.

3 Backend

A highly modularized system has been created and will be described module by module. There are two module categories, frontend and backend (surprise) with gpgme somewhat in between.

3.1 GPG for S/MIME

gpgsm is the S/MIME counterpart to the gpg command line tool available for most Unix systems. The look and feel is pretty much the same except that there is no verbose `--edit` command with its dozens of subcommands. Due to the way S/MIME is organized, there is no need for such advanced key management. There is just a generate key command and a couple of commands to maintain the database of certificates.

Private keys are not handled by gpgsm instead the operations of signing and decryption are delegated to the GpgAgent. When decrypting, this delegation only concerns the decryption of the session key; the symmetric decryption however is done by gpgsm. For signing only the hash is passed on but it is planned to optionally allow to send the entire document which makes sense if the GpgAgent runs on a more trusted machine.

The program is capable of encrypting data streams of arbitrary length without consuming too much memory (well, if you choose to encode a 1 hour video into a DN you may run into problems).

gpgsm is also responsible for the generation of keys and the creation of certificate requests. The actual key generation will be delegated as usual to the GpgAgent, enabling it to save the private key directly in its PSE or to store it on a smartcard.

gpgsm utilizes 2 other programs: The DirMngr and the GpgAgent, for example a signature verification is done this way:

1. Check that the signature is valid
2. Check that the certificate is valid
3. Check that the root certificate is in the list of trusted certificates; this is done by a calling GpgAgent with the `ISTRUSTED` command.
4. Check that that each certificate in the chain has not been revoked; this is done by calling DirMngr using the command `ISVALID`.

To access the certificate, GPGSM stores them in a database called the keybox. This is a unified key storage which can keep both, X.509 certificates and OpenPGP keys. The entries are stored along with meta information like fingerprint and stringified DNs to make lookup operations fast.

3.2 KSBA

Although KSBA should be pronounced as “Kasbah”, it should not be decoded as the inner city of an Arabian town but using rot-13 and replacing any digits with the first letter of their name.

It is mainly a library used to provide a standard API to X.509 and CMS[2] data structures. It takes a hybrid implementation approach of table driven BER decoder and manually crafted parser parts to gain good performance. The use is pretty straightforward and documented with examples in a manual. KSBA can of course work with streamed data and provides a couple of subsystems:

- X.509 certificate parser
- CMS parser and builder
- CRL parser
- PKCS-10 builder
- DN name transformations
- OID helper
- Writer and reader objects

More functionality will be added as needed. We are not aiming for CA software so a PKCS-10[4] parser and CRL build functions are not available. This may of course change in the future.

The subsystems share a common way of usage: `ksba_foo_new` creates a new object for a subsystem (e.g. cert, cms, crl) which in turn will be passed to all other functions. A `ksba_foo_release` frees the object. Reference counting is used for some of the objects to avoid unnecessary copy operations. Complex data structures are either returned in a stringfied version (e.g. distinguished names are converted according to RFC-2253) or as S-Expressions[6] in canconical representation.

The structure of the CMS and CRL parsers and build functions are implemented without callbacks by using a looping construct like:

```
do {
  ksba_foo_parse (obj, &stopreason);
  switch (stopreason)
    {
      case KSBA_SR_NEED_HASH: setup_hashing (obj); break;
      case KSBA_SR_GOT_SIG: verify_signature (obj); break;
      default: break;
    }
} while (stopreason != KSBA_SR_READY);
```

This enables us to change the internal working of LIBKSBA without requiring changes in an application program. New stop reasons can be introduced to allow for more fine graded control of the parser.

3.3 GPG Agent

The GpgAgent is responsible for all private key management. Private keys are help inside of the agent and operations employing it are never done by a application program but solely inside of GpgAgent. This will allow us to eventually implement a GpgAgent compatible software on a handheld device, so that we can get the best possible protection for secret keys and be sure that signed documents can be viewed on a trusted device before signing.

GpgAgent is also responsible to delegate crypto requests to the SCdaemon.

Another important piece of information are the trusted keys (CA-Root keys or ultimately trusted keys for OpenPGP). GpgAgent manages them and provides a simple API to access them. To insert a new trusted key into this list, GpgAgent will pop up a window to present information on that key.

The usual way to use GpgAgent is to start it up by the login shell and set a shell variable with information needed to access the agent (i.e. the name of the socket). This is as easy as `eval ‘gpg-agent‘`. Because GpgAgent caches passphrases in memory this mode of operation is preferred over executing it as `gpg-agent --server` which is done by clients if no GpgAgent has been started.

3.4 SCdaemon

The smartcard daemon takes care of accessing smartcards. It is separate from GpgAgent due to different linkage requirements (there is no need for LIBKSBA in GpgAgent but we need to parse X.509 data structures when handling smartcards) and for the benefit of applications not needing to access GpgAgent (e.g. tools to list the content of a smartcard or using non-confidential information).

The interface is the usual Assuan style one which enables easy testing and simple use from all kind of programming languages without the problems of library dependencies.

The daemon does support PKCS-15 cards and comes with a tool to create a PCKS-15 file system on a blank card. A `LEARN` command can be used by applications to get an overview of an inserted card. GpgAgent does use this command to register a new card with its internal database of private keys and trusted certificates; GPGSM is able to retrieve certificates (via GpgAgent) from the smartcard by means of this command. Eventually support for non PKCS-15 cards will be integrated so that German SigG/SigV compliant cards may be used.

3.5 GnuPG Made Easy

GPGME is an application library used to integrate GnuPG with other applications. It will provide language bindings for a couple of common languages as well as support for common component models.

GPGME provides an abstract API which is not fixed to any encryption protocol. The current version provides support for the backends GPG and GPGSM. It is quite convenient to use this library because the actual way to interface with GPG is not easy to implement and GPGSM even uses a different interface.

The OO approach used by GPGME makes integrating of S/MIME or OpenPGP into a MUA or other applications really easy. Currently we don't support streaming operations, however this is just an implementation issue, the API has all the functions to allow this. Note, that all mailers I am aware of, are keeping the messages in memory so that there is no need to care about this for now.

3.6 Directory Manager

The DirMngr is designed as a system service to manage certificate revocation lists (CRLs) as well as to provide an interface for address books to access LDAP directory services.

CRLs are an integral part of the S/MIME infrastructure and essential for encrypting and verification of messages. Because CRLs can get quite huge and it would be a privacy as well as performance problem to check a directory service each time a message is encrypted or verified, the DirMngr caches CRLs and can decide whether the currently cached entries need a refresh which it will then do.

As of now, the DirMngr is employed by GPGSM for each recipient certificate used in an encryption operation as well as for all signer certificates of a signed message. The interface used is pretty simple: The client sends a request of the form:

```
ISVALID 01234567890ABCDEF01234567890ABCDEF.013456
```

and DirMngr checks whether it has a recent CRL for the issuer CA identified by the hash of the issuers DN and then looks up the entry with the serial number given after the dot. The response is a short

```
OK
```

or an error where the error number will usually be

```
ERR 301 Certificate Revoked
```

If there is a problem retrieving a current CRL other error messages are returned.

The DirMngr will eventually be extended to retrieve CRLs using protocols other that LDAP, like HTTP or FTP.

4 Frontend

With that backend we have all what we want. But wait, there are users and they rightfully demand an interface for their daily use as non hackers. So let's see what we can do for them.

The GNUS users are not the audience for this, Gnus will make direct use of the backend engines GPG and GPGSM. The EVOLUTION folks don't like GPGME because they want to support the proprietary PGP software.

We have selected two MUAs to be enabled with the full *Sphinx* functionality. In addition SYLPHEED, which used to be the test bed for GPGME, will be enhanced with S/MIME support automagically by using GPGME.

4.1 KMail

KDE comes with a blinking and whistling colorful thing called KMAIL. Our customer required KMAIL support, so we did it.

As it turned out, the job was harder than even assumed by the KDE folks due to lack of a real MIME framework. We had to provide all of this. KMAIL does now have a crypto plugin interface which can be used to load any protocol engine one desires (as long as it conforms to our interface description ;-). We have removed the old PGP code from KMAIL and moved it into such a crypto plugin; this was needed to please some of the KMAIL developers. For our purposes we provide 2 plugins which are nearly identical: one for S/MIME and one for OpenPGP.

4.2 Mutt

Folks who are not using Gnus, are very likely using Mutt. We are going to replace the current PGP support in Mutt using our new system. This will be a compile time option and time will tell what interface users do prefer. Given that Mutt is a kind of reference implementation for MIME, it should be straightforward to add *Sphinx* support.

4.3 Sylpheed

SYLPHEED is a small footprint graphical MUA with a lot of useful features. Due to the fact that GPGME has been used for PGP support since the beginning, S/MIME support is practically for free.

4.4 PIN Entries

We have developed two PIN entry modules which are used by GPGAGENT to ask for PINs and passphrases. Extra caution has been taken to make them as secure as possible. For example the GPINENTRY program uses a GTK+ text entry widget, especially developed never to store the entered string in memory parts which may get paged out.
The programs are:

- KPINENTRY is an application with the look and feel of the KDE desktop and best suitable when using the KDE.
- GPINENTRY is a lightweight GTK+ application which is useful if KDE is not used.

both programs provide the same interface and pop up in the middle of the screen with grabbed keyboard and mouse to avoid X event sniffing. It is planned to write another one which does not need X at all but utilizes NCURSES or the Linux VCS devices to pop up a window.

5 Conclusion

We hope that the project will foster the use of encryption technology on GNU/Linux and other POSIX conforming operating systems and eventually establish a standard framework when used as desktop systems.

My thanks belong to the hackers working on this project and all the volunteers giving us valuable input.

A Online Resources

Further information including the status of the project and how to access the CVS is available at:
`http://www.gnupg.org/aegypten/`

References

[1] J. Callas et al. RFC 2440: OpenPGP message format, November 1998.

[2] R. Housley. RFC 2630: Cryptographic message syntax, June 1999.

[3] RSA Laboratories. PKCS #11 v2.11 final draft: Cryptographic token interface standard, jun 2001.

[4] M. Nystrom and B. Kaliski. RFC 2986 PKCS #10: Certification request syntax specification, November 2000.

[5] B. Ramsdell. RFC 2633: S/MIME version 3 message specification, June 1999.

[6] Ron Rivest. Code and description of s-expressions (http://theory.lcs.mit.edu/ rivest/sexp.html).