Netpgp - BSD-licensed Privacy Software

Alistair Crooks
The NetBSD Foundation
agc@NetBSD.org
d415 9deb 336d e4cc cdfa 00cd 1b68 dcfc c059 6823
August 2009

Abstract

This paper describes libnetpgp, which is a library built to provide privacy functions (digital signing, verification, encryption and decryption). This library has a BSD licence. On top of the libnetpgp library, a number of utilities are crafted, all dealing with different aspects of key management, data manipulation, and signed data verification. This software is unique in a number of ways - it can be embedded in BSD-licensed software with no licensing problems, it is designed to be able to handle large files and data streams, and the user-interface aspects have been augmented to provide an improved user experience. This paper describes the library and its functionality, shows how it can be used in practice, and then outlines areas for future study.

1. Background

PGP was written in 1991 [Zimmerman1991], and was instrumental in challenging the US laws on encryption and munitions. A GNU variant, *gnupg* or *gpg*, [GPG2009] provides some of the functionality which *pgp* provides, but presents the user with a bewildering array of command line options, to the extent that gpg has been described to the author as "the utility with the worst user interface ever". gpg is available with a library interface, *gpgme*, so that *gpg* functionality can be embedded in other programs and libraries. More information can be found in [Wikipedia2009], under the section marked "Problems"; this text explains how gpgme is simply a fork and exec wrapper around the gpg standalone utility, which can present its own set of problems.

In 2005, there was a Google Summer of Code project for the NetBSD project to provide a BSD-licensed Privacy Guard, known as BPG. The project was successful, in that it allowed files to be signed with DSA signatures, and to be verified. Provision of RSA or Elgamal signatures was beyond the scope of the project, as was key management; likewise encryption and decryption. In retrospect, the acronym chosen for its name was a mistake, since it is often mistaken with BGP (the Border Gateway Protocol), with which it shares no common features whatsoever. The BPG sources can be found in NetBSD's *othersrc* repository.

At the end of 2008, Ben Laurie made the *openpgpsdk* software available, constructed on top of the big number functionality provided by openssl. This library was the corollary of openssl in a number of ways - it was fully-provisioned, and did everything it could. There are a huge number of functions exported by openpgpsdk. User-level access was by means of a library, and a single utility providing command-line access. There were also a number of deficiencies - the verification and decryption of files larger than 8192 bytes was not supported. At the same time, the author of this paper found some of the constructs difficult to understand and use, and a single, small, high-level interface to the openpgpsdk library was needed. From that requirement, libnetpgp(3) was born, along

with a netpgp(1) driver program. Eventually, a number of userland utilities also emerged:

- netpgp(1) deals with signing, verification, encryption and decryption.
- netpgpkeys(1) handles key management.
- netpgpverify(1) is a single, standalone utility which performed only verification of files.

This paper gives a brief overview of public and private keys, the web of trust and looks at openpgpsdk and the reasons for netpgp's existence; and describes the user interfaces; differentiators of the two systems are listed, and future work is identified.

2. Netpgp and Trust

2.1 Web of Trust

A key identifying a person is split into two separate parts - a public and secret part. In fact, this key is a large multiple precision number (or "big" number), which is the product of two prime numbers. The size of this number is decided at key generation time. Care should be taken when generating online identities. Previous recommendations for key length usually advise that they key length should be 1024 bits. This is now considered to be too small. At the time of writing (late 2009), 4096 bits should be considered as the minimum possible key length, with 8192 bits as recommended. By default, gpg places the public part of the key in a user's pubring.gpg file in the $HOME/.gnupg directory. Other public keys are also kept in that same file. A user's public key can be uploaded to any of a number of public key servers.

The secret part of the key (or secret key) is protected by a password. The secret key is placed by default by gpg in the secring.gpg file in the user's $HOME/.gnupg directory.

The key can be identified by a number of different means: the key fingerprint, the userid (which is the last 8 or 16 characters of the fingerprint), the name (which does not identify a key uniquely), an email address (which identifies a specific identity, but is insufficient for identifying a key uniquely, since one person can have a number of separate keys, and because generating a key with a specific email address is easy - it is the trust of others in that key which identifies an identity uniquely). It is for this reason that a number of projects and groups get together to have key-signing parties.

The netpgp suite of programs uses the existing gpg pubring.gpg and secring.gpg files (by default, these are located in the .gnupg subdirectory of a user's home directory). The pubring.gpg and secring.gpg files are created at key generation time.

There is also a secret keyring (usually containing a single key, and a number of subkeys). These make up the secret part of the user's identity.

2.2 Signing and verification

Given some data, the secret key is used to "sign" that data. Anyone who has the public key can "verify" that the same data has not been modified. The way this is done is that a message digest of the data is taken. That digest is then modified by the user using the secret key, and the result is the signature.

The data itself can be verified that it has not been changed - either maliciously or by

failure of a component. A one-bit change in the data will mean that the computed signature at verification time will not match the signature as distributed.

The signature can be appended to the original file for ease of use - copying, transfer between machines, etc. Alternatively, it can be manipulated as a detached signature, in a file separate to the original file.

In addition, the signature can be "ASCII armoured". This means that the binary data of the signature is base64-encoded.

A digital signature not only verifies the data itself - it also provides information about the signatory, and the date and time that the signature was made.

This can be used in email to verify that, with so many spoofing measures available in our mail user agents, the mail really was sent by the owner of the private key, and the date and time at which it was sent.

2.3 Encryption and decryption

Netpgp encryption can be thought of as being the inverse operation to signing. Instead of using the signatory's private key to produce the signature, the intended recipient's public key is used. The recipient, and only the recipient, is then able to read the data using their private key to decrypt the information. Since this key is kept secret, others are unable to read the data.

This is now a de-facto standard for data backup off-site. The encryption of data means that even if media is lost or mislaid, the recipient will be unable to view the data (without the owner's private key). It can also be used to protect data after media has been recycled. Again, the lack of the owner's private key means that the data cannot be read.

In email, we encrypt using the recipient's public key to make sure that only the intended recipient can view the message. This is used extensively in the security community these days.

2.4 Standards

RFC 4880 [IETF2007] is the standard which manages openpgp implementations - it actually defines the openpgp message format. Because openpgpsdk was written to conform to this standard, netpgp is also compliant.

2.5 Key Management

With public and private keys in daily use, some means of distributing public keys is needed - for this, we "piggyback" on the existing structure that exists for PGP keys. This is seamless, since our keys are indistinguishable from PGP or GPG keys.

3. Openpgpsdk and Netpgp

Openpgpsdk was written with different goals - from the literature distributed with it, its main goal is to allow people to write modules to link with it to verify other signature types. All of its functions, low-level as well as high-level, are exported in a manner similar to openssl. Some shortcomings of openpgp were also identified - the ability to

verify a signature only if the original data was less than 8192 bytes in size, and the ability to decrypt of less than 8192 bytes, were considered critical, and were one of the first areas to be addressed.

Openpgpsdk exports around 9 header files - numerous structs and functions, both at the higher and lower levels of the openpgpsdk library. Netpgp has a single *netpgp.h* header file, which defines all of the high level functions (and none of the lower level ones), and a single `netpgp_t` structure, which is used to manage all of the state information.

Netpgp itself is built on top of openpgpsdk. All of the openpgpsdk functions have been "hidden", and the only exported functions are high level functions. Similarly, the `netpgp_t` struct is used by netpgp library functions use that as the handle for its instances. No static data is kept in netpgp.

4. Netpgp

4.1 User Interface

The netpgp command line interface has differed from the openpgpsdk and the gpg interfaces. This was one of the primary driving factors around the development of netpgp. For the command line, there are three programs:

1. netpgp - for signing and verifying data, and for encryption and decryption of data
2. netpgpkey - for all key management
3. netpgpverify - a standalone utility for simple verification of signed data

With the split between netpgp data and key management, a lot of the confusion about command line arguments went away. For instance, an invocation of netpgp to sign some data needs to know about hash algorithm choices, but not the favourite key server. Certainly, the manual pages were easier to write.

In the same way, a high-level interface has been provided for the library interface. The header file includes prototypes for the following functions:

```
/* begin and end */
int netpgp_init(netpgp_t *);
int netpgp_end(netpgp_t *);

/* debugging, reflection and information */
int netpgp_set_debug(const char *);
int netpgp_get_debug(const char *);
const char *netpgp_get_info(const char *);
int netpgp_list_packets(netpgp_t *, char *, int, char *);

/* variables */
int netpgp_setvar(netpgp_t *, const char *, const char *);
char *netpgp_getvar(netpgp_t *, const char *);

/* key management */
int netpgp_list_keys(netpgp_t *);
int netpgp_find_key(netpgp_t *, char *);
int netpgp_export_key(netpgp_t *, char *);
```

```
int netpgp_import_key(netpgp_t *, char *);
int netpgp_generate_key(netpgp_t *, char *, int);

/* file management */
int netpgp_encrypt_file(netpgp_t *, const char *, const char *, char *,
int);
int netpgp_decrypt_file(netpgp_t *, const char *, char *, int);
int netpgp_sign_file(netpgp_t *, const char *, const char *, char *, int,
int, int);
int netpgp_verify_file(netpgp_t *, const char *, const char *, int);
```

which are the only way to interface to libnetpgp(3).

4.2 Differentiators

The netpgp suite joins a long line of public/private key programs and libraries, and so perhaps the reasons for its existence are unclear.

- Firstly, gpg and gpgme are covered by the GPL, and so are unsuitable for embedding in a BSD-licensed piece of software. In addition, gpgme is not a library implementation per se - it carries out work by using fork(2) and exec(2) from within the library to call gpg(1) on the command line. This has problems for threaded programs - since the behaviour of a fork(2) in a program which has already spawned threads may be undefined. So embedding gpgme not only has performance problems, but may be unpredictable in operation depending on the calling site. By contrast, libnetpgp has none of these problems - all state data is held in the netpgp_t structure which is used in every library API call, and no external processes are used to get or set data.

- As mentioned before, openpgpsdk has problems with verification and decryption of files larger than 8192 bytes (this failure mode is, unfortunately, silent). In order to work with files of arbitrary size, netpgp is needed.

- As a performance boost, netpgp uses an mmap(2) interface to read the data. If this fails, netpgp falls back to read(2), which is usually guaranteed to work, as long as the user has permission to read that data.

- netpgp is fully GNU autoconf-ed. openpgpsdk has a custom configure script written in Perl, and relies on Cunit to be present even if no testing is to be done. In addition, netpgp has been extended to use libtool, and to have autotest functionality.

- Userids (the last 8 or 16 hexadecimal bytes of the key fingerprint) are now matched in a case-insensitive manner to improve the user experience. Matching can also be done on email address, although this is not encouraged, since email addresses in keys are not authenticated in any way - that is, anyone can generate a key with the address <agc@netbsd.org> in it. However, the c0596823 userid together with the address <agc@netbsd.org> is unique.

- Library user experience - netpgp has moved to text-based parameters. This improves the interface for calling programs, and also simplifies calling from other language bindings.

- Manual pages have been written for the library and the utilities.

- The code is all WARNS=4 safe. This means that the code compiles with "gcc -Werror -Wall" plus other warnings.

- All *assert*() calls have been removed - they are placed with runtime checks, and errors thrown or returned. This caused some controversy on the NetBSD mailing lists, but should be reviewed in light of the recent bind zero-day vulnerability (which was due to an assertion being triggered by a specially-crafted dynamic update message).

- Core dumps have been disabled by default; they can be enabled by specifying the --coredump option on the command line. The reason for this is because sensitive data (the secret key, and possibly its protecting password) may be held in memory, and thus dumped to disk when a core dump is taken. This feature ensures that, by default, no sensitive data ends up on disk unnecessarily.

- By default, SHA256 digests of files are used, rather than SHA1. This is because collisions in sha1 are becoming more common, and easier (although still relatively difficult) to generate. To try to protect ourselves as much as we can, and since the whole signature of a document relies on the digest, netpgp moved to SHA256 as the default digest type.

- netpgpkeys now prints out the size of the key in list-keys, to mirror the key display provided by gpg.

- netpgp also prints out the key information before prompting for any password for the decryption key. This is in case a number of keys are in use by a single user, so that the user can identify which key is desired.

- There was an asymmetry in the functionality originally provided by netpgp. When signing a file, the signature is appended to the file itself - this is the default mode. Verification of that signature will return a boolean answer of whether the signature is a good one or not. However, there was no way to retrieve that original data - remember that the act of signing it appended the signature to the file. The --cat command verifies a signed file, and, if the signature is good, displays the original contents of the file without its associated signature.

- netpgp has a separate standalone verification program, and a separate key management program. This is partly to address the concerns about command line confusion which was explored earlier. It is also a recognition of the fact that these programs do separate things, and should not automatically be lumped together into one.

- C++ guards have been provided in the *netpgp.h* header file.

- --passfd was added to netpgp(1), which is used as the file descriptor on which the password is provided to the calling program. Care and discretion should be used with this

4.3 Implications

With netpgp we now have a user-level library and standalone utilities which can be used to verify the provenance of arbitrary data on our systems. In that way, netpgp can be thought of as checksums or message digests on steroids. We no longer have to protect these checksums, though, since the date and identity of the signatory can be verified.

That means that digital signatures could be used in a number of places on our systems where we current use message digests.

An example of this is in our packaging systems, where we use two message digests, SHA1 and RMD160, to verify distfiles; we also use individual (MD5) message digests to verify individual files in the inventory list of installed packages. Other packaging systems (such as rpm) use a digital signature on the inventory file itself. If our inventory files were augmented to keep owner, group and mode information, then we could use a single digital signature on the inventory file itself to ensure that it has not been modified in any way.

Manufacturers and vendors use digital signatures to ensure that authorised applications are being run on their equipment. A number of ways of doing this are available.

NetBSD has a kernel subsystem called veriexec, which ensures that a binary's calculated checksum matches a pre-computed one (loaded at boot time while the kernel is in a secure state). By reworking the loading of checksums to load trusted signatory public keys, a binary's digital signature can be verified at execution time - if the signatory matches one of the trusted public keys, then the binary can be executed. To do this, verification of signatures needs to be available in the kernel.

5. Worked Example

netpgp can be used to sign a file:

        % netpgp --sign netpgp-2009.txt
        netpgp: default key set to "c0596823"
        pub 2048/RSA (Encrypt or Sign) 1b68dcfcc0596823 2004-01-12
        Key fingerprint: d415 9deb 336d e4cc cdfa 00cd 1b68 dcfc c059 6823
        uid   Alistair Crooks <agc@netbsd.org>
        uid   Alistair Crooks <agc@pkgsrc.org>
        uid   Alistair Crooks <agc@alistaircrooks.com>
        uid   Alistair Crooks <alistair@hockley-crooks.com>
        netpgp passphrase:
        Bad passphrase
        pub 2048/RSA (Encrypt or Sign) 1b68dcfcc0596823 2004-01-12
        Key fingerprint: d415 9deb 336d e4cc cdfa 00cd 1b68 dcfc c059 6823
        uid   Alistair Crooks <agc@netbsd.org>
        uid   Alistair Crooks <agc@pkgsrc.org>
        uid   Alistair Crooks <agc@alistaircrooks.com>
        uid   Alistair Crooks <alistair@hockley-crooks.com>
        netpgp passphrase:
        % ls -al netpgp-2009.txt*
        -rw-r--r--@ 1 agcrooks staff 13894 30 Aug 11:18 netpgp-2009.txt
        -rw------- 1 agcrooks staff 14208 30 Aug 18:57 netpgp-2009.txt.gpg
        %

and the same signed file can be verified by both netpgp and by gpg:

        % netpgp --verify netpgp-2009.txt.gpg
        netpgp: default key set to "c0596823"
        Good signature for netpgp-2009.txt.gpg made Sun Aug 30 18:57:05 2009
        using RSA (Encrypt or Sign) key 1b68dcfcc0596823
        pub 2048/RSA (Encrypt or Sign) 1b68dcfcc0596823 2004-01-12
        Key fingerprint: d415 9deb 336d e4cc cdfa 00cd 1b68 dcfc c059 6823
        uid   Alistair Crooks <alistair@hockley-crooks.com>

```
uid              Alistair Crooks <agc@pkgsrc.org>
uid              Alistair Crooks <agc@netbsd.org>
uid              Alistair Crooks <agc@alistaircrooks.com>
% gpg --verify netpgp-2009.txt.gpg
gpg: Signature made Sun 30 Aug 18:57:05 2009 PDT using RSA key ID C0596823
gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, classic trust model
gpg: depth: 0  valid:   1  signed:  35  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: depth: 1  valid:  35  signed:  33  trust: 2-, 1q, 2n, 27m, 3f, 0u
gpg: depth: 2  valid:  14  signed:  15  trust: 0-, 3q, 3n, 6m, 2f, 0u
gpg: next trustdb check due at 2009-12-21
gpg: Good signature from "Alistair Crooks <agc@pkgsrc.org>"
gpg:              aka "Alistair Crooks <alistair@hockley-crooks.com>"
gpg:              aka "Alistair Crooks <agc@netbsd.org>"
gpg:              aka "Alistair Crooks <agc@alistaircrooks.com>"
%
```

6. Future Work

A number of significant enhancements to netpgp have been identified. These are:

1. The intention is to separate netpgpverify from openssl by providing the crypto and mpi functions separately, and to wrap the kernel's AES/Rijndael functions in an openssl-compatible API. netpgp can then be "dropped" into the kernel. This will give us the ability to then hook netpgp verification of files into the veriexec system, as described earlier. veriexec "digests" can be loaded from userland as the detached signatures of the binary files. Trusted public keys can be loaded at early-boot securelevels. netpgp verification of binary files can then be performed at binary execution time, which will be slower than normal, but provide much better protection for small, embedded devices, or bastion hosts. This work is underway, and it is estimated that the code will hit the NetBSD repository before the end of October 2009. It should be noted that signed binary execution works well for static binaries. For dynamic binaries, some form of integration into ld.elf_so is necessary.
2. netpgpkeys needs some work to enable key management to be performed. At the present time, the emphasis has been placed on netpgp working with data - digital signature and verification, encryption and decryption have been the focus. As netpgp works with the pubring.gpg and secring.gpg files produced by gpg, that utility is used right now for key management. Replacing this is a priority in netpgp development.
3. Bulk-signing of data is also a high priority development item at the time of writing.
4. By default, gpg will gzip the original data (if it is file-based) when signing the data. netpgp, by contrast, does not gzip the data. In light of recent vulnerabilities in the zlib and bzip2 libraries, this may be regarded as being fortunate. However, the principle of least astonishment calls for default gzip compression of files signed with netpgp.
5. Another area for development is ElGamal encryption and decryption, for use with DSA signatures.
6. gpg allows the encryptor to specify multiple decryption user ids - the resulting encrypted file can be decrypted by any of the private keys by which it was encrypted. netpgp will follow a similar, but more expressive and dynamic system based on threshold schemes such as Shamir's Secret Sharing Scheme [SSSS2005]. A threshold scheme allows a secret to be shared amongst a number of people, and the secret can be revealed only when the pre-defined threshold (minimum number) of shares of the secret have been obtained.

Possible application areas for this are decryption of encrypted backups by a team of backup specialists, and data escrow, to name but two.
7. Foreign language bindings - lua bindings have been written for libnetpgp, as have Python bindings (by Oliver Gould). Perl bindings are a work in progress.

7. Conclusion

netpgp is the result of a substantial amount of work, and would not have been possible without openpgpsdk. Our thanks are due to its authors, Rachael Wilmer and Ben Laurie. netpgp has different aims and goals to openpgpsdk, though, and its differentiators can be shown to be necessary in a signing and verification, and encryption and decryption library and utility for the BSD world. It is suitable for embedding, and can verify signatures made by gpg, and decrypt files which were encrypted by gpg. The keys and key files used are interchangeable with gpg; yet libnetpgp suffers from none of the fork and exec drawbacks of the equivalent gpgme library, in addition to netpgp's BSD licensing. There are a number of areas where we currently use digests, where digital signatures could be used more efficiently and to better effect.

8. References

[GPG2009]   http://www.gnupg.org/
[IETF2007]   http://tools.ietf.org/html/rfc4880
[SSSS2005]   http://point-at-infinity.org/ssss/
[Wikipedia2009]   http://en.wikipedia.org/wiki/GNU_Privacy_Guard
[Zimmerman1991]   http://philzimmermann.com/EN/essays/index.html