NetPGP
BSD-licensed Privacy

Alistair Crooks
agc@NetBSD.org
c059 6823
Privacy?

- Encryption and decryption
- Signing and verification
- Web of trust
- PKI
- Certifying Authority
Today?

- pgp
- gnupg
- gpgme
- proprietary privacy software
Requirements

• BSD-licensed
• Embeddable
• Performant
• Command line invocation without Ph.D.
Cats: All your base are belong to us.
Base

- openpgpsdk library - Ben Laurie & Rachel Wilmer
- decrypts files \(\leq 8192\) bytes
- verifies files \(\leq 8192\) bytes
- no detached signing or verification
OpenPGPSDK Structure

- 12 header files
- Masses of structs, functions, definitions
- No namespace limits or cues
- Large identifier names
NetPGP

- Layer above OpenPGPSDK
- Hides openpgpsdk implementation
- One header file
- One structure used for state
- Configuration values by (key, value) strings
Goals

- BSD-licensed embeddable library
- GPG compatibility
What is PGP?

- Based around IETF RFC 4880
- describes PGP message format
- obsoletes RFC 1991, 2440
How does it work?

- Two parts to key
  - public
  - private
PGP Signing and Verification

- File is signed with secret key of signer
- Can be verified with the public key of signer
- Proves provenance of file or stops file contents being disclosed
Problems? Challenges?

- “I can’t use gpg”
- I have problems too
- It says more about gpg than me, though
- “Mail is OK, but everything else is hard”
- Yes
PGP Encryption & Decryption

- File is encrypted with the public key
- Can only be decrypted with the secret key
- Corollary/complement of signing
Differentiators

- Why would I use netpgp over gpg?
Licensing

• BSD licensing
• Library can be used in places other libraries can’t reach
• Practical, usable signing and verification
• Encryption and decryption
GPG compatible

- Use existing ~/.gnupg directory
- Use existing keys and configuration
- Use existing infrastructure for public keys
- Use files signed/encrypted by gpg
Library

- gpgme has problems
- embeddable anywhere
- could be added to existing programs
Language bindings

- Python
- Perl
- Lua
- C/C++
Split by functionality

• One binary for signing, verification, encryption and decryption
• One for key management
• Standalone binary for signature verification
Nantucket
Netpgp in Action

- Taken from portable regression test suite
- GNU auto tools tests not shown here
Sign 748579 byte file

% /usr/bin/netpgp --sign a
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:
%
Verify that Signature

% /usr/bin/netpgp --verify a.gpg
netpgp: default key set to "C0596823"
Good signature for a.gpg made Sat Sep 19 06:52:46 2009
using RSA (Encrypt or Sign) key 1b68dcfccc0596823
pub 2048/RSA (Encrypt or Sign) 1b68dcfccc0596823 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>
%
Encrypt 13501 byte file

% /usr/bin/netpgp --encrypt b
netpgp: default key set to "C0596823"
%
Decrypt that file

```bash
% /usr/bin/netpgp --decrypt b.gpg
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:
%```
% /usr/bin/netpgp --sign --detached f

netpgp: default key set to "C0596823"

pub 2048/RSA (Encrypt or Sign) 1b68dcfccc0596823 2004-01-12
Key fingerprint: d415 9deb 336d e4cc cdfe 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 -l f f.sig

-rw-r--r-- 1 agc agc 4491474 Sep 19 06:53 f
-rw-r--r-- 1 agc agc  287 Sep 19 06:53 f.sig
%

%
% /usr/bin/netpgp --verify f.sig
netpgp: default key set to "C0596823"
netpgp: assuming signed data in "f"
Good signature for f.sig made Sat Sep 19 06:53:06 2009
using RSA (Encrypt or Sign) key 1b68dcfccc0596823
pub 2048/RSA (Encrypt or Sign) 1b68dcfccc0596823 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>
%
% '/usr/bin/netpgp' '--cat' 'a.gpg'

netpgp: default key set to "C0596823"

Good signature for a.gpg made Sat Sep 19 06:52:46 2009 using RSA (Encrypt or Sign) key 1b68dcfccc0596823
pub 2048/RSA (Encrypt or Sign) 1b68dcfccc0596823 2004-01-12
Key fingerprint: d415 9deb 336d e4cc cdff 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>

%
Message digests

- Digital signatures are simply manipulation of a digest of a file using the secret key
- Problem with digest collisions
- Use of SHA2 instead of SHA1 for digests
Use Cases
Signed email

- Sign email with a digest - everyone can verify sender by using sender’s public key
- PGP public keys available on key servers
- Compatible with gpg
  - except for the user interface
Signed software releases

- Projects usually sign the major releases
- People can verify that they have an official version
Signed Binaries

- Some embedded manufacturers use signed binaries to show provenance of binary
- Kernel will only execute a binary if it is known to come from reliable source
Signed backups

• To make sure that backups are good, they can be signed

• If anything changes, can be flagged

• Proves that backup was valid at one point in time
Well...

- Does a single signature prove the backup is good?
Multiple signatures

- Nested, or
- Detached signatures
Detached signatures

- Allow a file to remain unchanged
- Signature is in a separate file
- Any number of detached signatures can be made
Embedded signatures

- Lack of symmetry
- Get contents of signed file
- After verification
Standalone Verification

- Separate program which verifies signature
Key Management

• Separate program which manages keys
• Not a priority
• Will list keys
• More development needed
Multiple encryption

- How do we do this for multiple people to decrypt?
- Backup tapes, for example
Indirection

- By using a key to encrypt the backup tape
- And then giving that key to multiple parties, encrypted with their own key
Byzantine refers to the Byzantine Generals' Problem, an agreement problem in which generals of the Byzantine Empire's army must decide unanimously whether to attack some enemy army. The problem is complicated by the geographic separation of the generals, who must communicate by sending messengers to each other, and by the presence of traitors amongst the generals. These traitors can act arbitrarily in order to achieve the following aims: trick some generals into attacking; force a decision that is not consistent with the generals' desires, e.g. forcing an attack when no general wished to attack; or confusing some generals to the point that they are unable to make up their minds. If the traitors succeed in any of these goals, any resulting attack is doomed, as only a concerted effort can result in victory.

Byzantine fault tolerance can be achieved if the loyal (non-faulty) generals have a unanimous agreement on their strategy. Note that if the source general is correct, all loyal generals must agree upon that value. Otherwise, the choice of strategy agreed upon is irrelevant.
A different way...

- Shamir’s secret sharing scheme
Secret Sharing

- Shamir’s Secret Sharing Scheme
- Multi-dimensional space
- Threshold schemes
- Byzantine fault tolerance
Noteworthy changes in version 1.4.10 (2009-09-02)

2048 bit RSA keys are now generated by default. The default hash algorithm preferences has changed to prefer SHA-256 over SHA-1.

2048 bit DSA keys are now generated to use a 256 bit hash algorithm
Benefits

- Don’t modify exec format
- Don’t modify binary
- Hang onto NetBSD’s veriexec infrastructure
NetBSD & signed binaries

- Want to load signatures from userland
- Verify in kernel (only public key needed)
- Modify veriexec framework to check verification of signature via pubkey
Thank you

Alistair Crooks
agc@NetBSD.org
c059 6823
Questions?

Alistair Crooks
agc@NetBSD.org
c059 6823