

# An Advanced Introduction to GnuPG

Neal H. Walfield  
neal@gnupg.org

RMLL, 6 July 2015

# Outline

OpenPGP

GnuPG's Architecture

Good Practices

Neat Tricks

# OpenPGP

- ▶ Data integrity service for messages and files
- ▶ Defined in RFC 4880
  - ▶ Published in 2007
- ▶ Focus
  - ▶ Message format
  - ▶ Message reading, writing and verification algorithms
  - ▶ Crypto algorithms to use and their parameters

# Trade-offs

- ▶ Good for data at rest
  - ▶ Need to be able to decrypt data in decades
    - ▶ OpenPGP is more like tar than http/smtp/xmpp
    - ▶ Consequence: Hard to phase out old algorithms
- ▶ No interaction between encryptor and decryptor
  - ▶ Can't negotiate parameters dynamically
  - ▶ No perfect forward secrecy

# Trade-offs

- ▶ Good for data at rest
  - ▶ Need to be able to decrypt data in decades
    - ▶ OpenPGP is more like tar than http/smtp/xmpp
    - ▶ Consequence: Hard to phase out old algorithms
- ▶ No interaction between encryptor and decryptor
  - ▶ Can't negotiate parameters dynamically
  - ▶ No perfect forward secrecy

# RFC 4880bis

- ▶ IETF working group developing a new revision
- ▶ Major Goals\*
  - ▶ Deprecate weak algorithms (MD5, SHA1, RIPEMD160, 3DES, IDEA)
  - ▶ Introduce new ECC curves (25519, curve448)
  - ▶ New key derivation function
- ▶ Planned release: July 2016

---

\* <http://wiki.gnupg.org/rfc4880bis>

# Message Format

- ▶ Packet-based
- ▶ Designed for unbuffered (single pass) processing
- ▶ 17 packet types
  - ▶ Symmetrically encrypted data
  - ▶ Public-key encrypted session key
  - ▶ Signature packet
  - ▶ Public-Key Packet
  - ▶ Public-Subkey Packet
  - ▶ Secret-Key Packet
  - ▶ User ID Packet
  - ▶ etc.

# OpenPGP Algorithms

- ▶ Encryption / Decryption
- ▶ Signatures
- ▶ Key derivation function (s2k, string to key)
- ▶ Encodings (ASCII armor)

# Encryption Algorithm

|                       |                       |                             |
|-----------------------|-----------------------|-----------------------------|
| $\text{Enc}_{r_1}(s)$ | $\text{Enc}_{r_2}(s)$ | $\text{Enc}_s(\text{data})$ |
|-----------------------|-----------------------|-----------------------------|

- ▶ **Generate** a random session key ( $s$ )
- ▶ For each recipient, **output**  $\text{Enc}_{r_i}(\text{session key})$
- ▶ **Output**  $\text{Enc}_s(\text{data})$
- ▶ Why a session key?
  - ▶ Symmetric crypto is fast
  - ▶ For  $N$  recipients, we encrypt plaintext once and session key  $N$  times

# Encryption Algorithm

|                       |                       |                             |
|-----------------------|-----------------------|-----------------------------|
| $\text{Enc}_{r_1}(s)$ | $\text{Enc}_{r_2}(s)$ | $\text{Enc}_s(\text{data})$ |
|-----------------------|-----------------------|-----------------------------|

- ▶ **Generate** a random session key ( $s$ )
- ▶ For each recipient, **output**  $\text{Enc}_{r_i}(\text{session key})$
- ▶ **Output**  $\text{Enc}_s(\text{data})$
- ▶ Why a session key?
  - ▶ Symmetric crypto is fast
  - ▶ For  $N$  recipients, we encrypt plaintext once and session key  $N$  times

# An Encrypted Message

```
$ echo -n foo | gpg2 -e -r 630052D9 -r 8E678210 | gpg2 --list-packets
# off=0 ctb=85 tag=1 hlen=3 plen=268
:pubkey enc packet: version 3, algo 1, keyid C2B819056C652598
data: [2047 bits]
# off=271 ctb=85 tag=1 hlen=3 plen=268
:pubkey enc packet: version 3, algo 1, keyid AE19DAC58E678210
data: [2047 bits]
# off=542 ctb=d2 tag=18 hlen=2 plen=56 new-ctb
:encrypted data packet:
length: 56
mdc_method: 2
# off=563 ctb=a3 tag=8 hlen=1 plen=0 indeterminate
:compressed packet: algo=1
# off=565 ctb=cb tag=11 hlen=2 plen=9 new-ctb
:literal data packet:
mode b (62), created 1435751184, name="",
raw data: 3 bytes
```

- ▶ Two recipients
  - ▶ Self and someone else
  - ▶ (Can always encrypt to some key using `encrypt -to` in `gpg.conf`)

# An Encrypted Message

```
$ echo -n foo | gpg2 -e -r 630052D9 -r 8E678210 | gpg2 --list-packets
# off=0 ctb=85 tag=1 hlen=3 plen=268
:pubkey enc packet: version 3, algo 1, keyid C2B819056C652598
data: [2047 bits]
# off=271 ctb=85 tag=1 hlen=3 plen=268
:pubkey enc packet: version 3, algo 1, keyid AE19DAC58E678210
data: [2047 bits]
# off=542 ctb=d2 tag=18 hlen=2 plen=56 new-ctb
:encrypted data packet:
length: 56
mdc_method: 2
# off=563 ctb=a3 tag=8 hlen=1 plen=0 indeterminate
:compressed packet: algo=1
# off=565 ctb=cb tag=11 hlen=2 plen=9 new-ctb
:literal data packet:
mode b (62), created 1435751184, name="",
raw data: 3 bytes
```

► 5 packets

# An Encrypted Message

```
$ echo -n foo | gpg2 -e -r 630052D9 -r 8E678210 | gpg2 --list-packets
# off=0 ctb=85 tag=1 hlen=3 plen=268
:pubkey enc packet: version 3, algo 1, keyid C2B819056C652598
data: [2047 bits]
# off=271 ctb=85 tag=1 hlen=3 plen=268
:pubkey enc packet: version 3, algo 1, keyid AE19DAC58E678210
data: [2047 bits]
# off=542 ctb=d2 tag=18 hlen=2 plen=56 new-ctb
:encrypted data packet:
length: 56
mdc_method: 2
# off=563 ctb=a3 tag=8 hlen=1 plen=0 indeterminate
:compressed packet: algo=1
# off=565 ctb=cb tag=11 hlen=2 plen=9 new-ctb
:literal data packet:
mode b (62), created 1435751184, name="",
raw data: 3 bytes
```

- ▶ off: offset within stream
- ▶ Header
  - ▶ ctb: packet header (“cipher type byte”)
  - ▶ tag: packet type
  - ▶ hlen, plen: header and payload length (in bytes)

# An Encrypted Message

```
$ echo -n foo | gpg2 -e -r 630052D9 -r 8E678210 | gpg2 --list-packets
# off=0 ctb=85 tag=1 hlen=3 plen=268
:pubkey enc packet: version 3, algo 1, keyid C2B819056C652598
data: [2047 bits]
# off=271 ctb=85 tag=1 hlen=3 plen=268
:pubkey enc packet: version 3, algo 1, keyid AE19DAC58E678210
data: [2047 bits]
# off=542 ctb=d2 tag=18 hlen=2 plen=56 new-ctb
:encrypted data packet:
length: 56
mdc_method: 2
# off=563 ctb=a3 tag=8 hlen=1 plen=0 indeterminate
:compressed packet: algo=1
# off=565 ctb=cb tag=11 hlen=2 plen=9 new-ctb
:literal data packet:
mode b (62), created 1435751184, name="",
raw data: 3 bytes
```

- ▶ pubkey enc packet
  - ▶ Encrypted session key
  - ▶ One for each recipient
  - ▶ Encrypted using the recipient's public key

# An Encrypted Message

```
$ echo -n foo | gpg2 -e -r 630052D9 -r 8E678210 | gpg2 --list-packets
# off=0 ctb=85 tag=1 hlen=3 plen=268
:pubkey enc packet: version 3, algo 1, keyid C2B819056C652598
data: [2047 bits]
# off=271 ctb=85 tag=1 hlen=3 plen=268
:pubkey enc packet: version 3, algo 1, keyid AE19DAC58E678210
data: [2047 bits]
# off=542 ctb=d2 tag=18 hlen=2 plen=56 new-ctb
:encrypted data packet:
length: 56
mdc_method: 2
# off=563 ctb=a3 tag=8 hlen=1 plen=0 indeterminate
:compressed packet: algo=1
# off=565 ctb=cb tag=11 hlen=2 plen=9 new-ctb
:literal data packet:
mode b (62), created 1435751184, name="",
raw data: 3 bytes
```

- ▶ encrypted data packet
  - ▶ Contains the data (encapsulated)
  - ▶ Encrypted using the session key
- ▶ compressed packet
  - ▶ Nested within the encrypted data packet
- ▶ literal data
  - ▶ Nested within the compressed packet data packet

# An Encrypted Message

```
$ echo -n foo | gpg2 -e -r 630052D9 -r 8E678210 | gpg2 --list-packets
# off=0 ctb=85 tag=1 hlen=3 plen=268
:pubkey enc packet: version 3, algo 1, keyid C2B819056C652598
data: [2047 bits]
# off=271 ctb=85 tag=1 hlen=3 plen=268
:pubkey enc packet: version 3, algo 1, keyid AE19DAC58E678210
data: [2047 bits]
# off=542 ctb=d2 tag=18 hlen=2 plen=56 new-ctb
:encrypted data packet:
length: 56
mdc_method: 2
# off=563 ctb=a3 tag=8 hlen=1 plen=0 indeterminate
:compressed packet: algo=1
# off=565 ctb=cb tag=11 hlen=2 plen=9 new-ctb
:literal data packet:
mode b (62), created 1435751184, name="",
raw data: 3 bytes
```

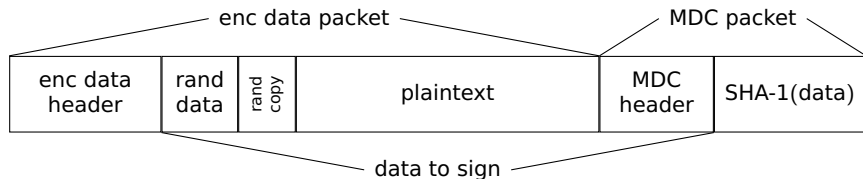
- ▶ Note the order of the packets
  - ▶ Encrypted Session key precedes encrypted data
  - ▶ No buffering needed to encrypt or decrypt data

# An Encrypted Message

```
$ echo -n foo | gpg2 -e -r 630052D9 -r 8E678210 | gpg2 --list-packets
# off=0 ctb=85 tag=1 hlen=3 plen=268
:pubkey enc packet: version 3, algo 1, keyid C2B819056C652598
data: [2047 bits]
# off=271 ctb=85 tag=1 hlen=3 plen=268
:pubkey enc packet: version 3, algo 1, keyid AE19DAC58E678210
data: [2047 bits]
# off=542 ctb=d2 tag=18 hlen=2 plen=56 new-ctb
:encrypted data packet:
length: 56
mdc_method: 2
# off=563 ctb=a3 tag=8 hlen=1 plen=0 indeterminate
:compressed packet: algo=1
# off=565 ctb=cb tag=11 hlen=2 plen=9 new-ctb
:literal data packet:
mode b (62), created 1435751184, name="",
raw data: 3 bytes
```

- ▶ Modification Detection System (MDS)
  - ▶ Integrity check
  - ▶ Normally done using signatures
  - ▶ But, signatures reveal the sender's identity
  - ▶ **MDS preserves integrity and anonymity**
- ▶ Modification Detection Code (MDC) packet
  - ▶ Immediately follows encrypted data packet
  - ▶ Not shown by gpg for technical reasons

# Modification Detection System



- ▶ **SHA-1 of:**
  - ▶ **Block of random data**
    - ▶ AES block size is 128 bits (16 bytes)
    - ▶ = 16 bytes of random data
  - ▶ **Last two bytes of random data are repeated**
    - ▶ Quick check for invalid key
    - ▶ No need to process TBs of data to check key
  - ▶ **The plaintext**
  - ▶ **Header of MDC Packet**
    - ▶ Included in hashed data
    - ▶ Detects data removal / extension attacks

# Signing Algorithm

- ▶ **Output** hash parameters
- ▶ Simultaneously **Hash** and **Output** message
- ▶ **Sign** hash using sender's private key
- ▶ **Output** signature

# A Signed Message

```
$ echo -n foo | gpg2 -s | gpg2 --list-packets
# off=0 ctb=a3 tag=8 hlen=1 plen=0 indeterminate
:compressed packet: algo=1
# off=2 ctb=90 tag=4 hlen=2 plen=13
:onepass_sig packet: keyid E149B3889E4DA08C
version 3, sigclass 0x00, digest 8, pubkey 1, last=1
# off=17 ctb=cb tag=11 hlen=2 plen=9 new-ctb
:literal data packet:
mode b (62), created 1435588610, name="",
raw data: 3 bytes
# off=28 ctb=89 tag=2 hlen=3 plen=284
:signature packet: algo 1, keyid E149B3889E4DA08C
version 4, created 1435588610, md5len 0, sigclass 0x00
digest algo 8, begin of digest 27 28
hashed subpkt 2 len 4 (sig created 2015-06-29)
subpkt 16 len 8 (issuer key ID E149B3889E4DA08C)
data: [2047 bits]
```

- 4 packets

# A Signed Message

```
$ echo -n foo | gpg2 -s | gpg2 --list-packets
# off=0 ctb=a3 tag=8 hlen=1 plen=0 indeterminate
:compressed packet: algo=1
# off=2 ctb=90 tag=4 hlen=2 plen=13
:onepass_sig packet: keyid E149B3889E4DA08C
version 3, sigclass 0x00, digest 8, pubkey 1, last=1
# off=17 ctb=cb tag=11 hlen=2 plen=9 new-ctb
:literal data packet:
mode b (62), created 1435588610, name="",
raw data: 3 bytes
# off=28 ctb=89 tag=2 hlen=3 plen=284
:signature packet: algo 1, keyid E149B3889E4DA08C
version 4, created 1435588610, md5len 0, sigclass 0x00
digest algo 8, begin of digest 27 28
hashed subpkt 2 len 4 (sig created 2015-06-29)
subpkt 16 len 8 (issuer key ID E149B3889E4DA08C)
data: [2047 bits]
```

- ▶ Compressed packet contains other packets
  - ▶ (Logical structure not shown)

# A Signed Message

```
$ echo -n foo | gpg2 -s | gpg2 --list-packets
# off=0 ctb=a3 tag=8 hlen=1 plen=0 indeterminate
:compressed packet: algo=1
# off=2 ctb=90 tag=4 hlen=2 plen=13
:onepass_sig packet: keyid E149B3889E4DA08C
version 3, sigclass 0x00, digest 8, pubkey 1, last=1
# off=17 ctb=cb tag=11 hlen=2 plen=9 new-ctb
:literal data packet:
mode b (62), created 1435588610, name="",
raw data: 3 bytes
# off=28 ctb=89 tag=2 hlen=3 plen=284
:signature packet: algo 1, keyid E149B3889E4DA08C
version 4, created 1435588610, md5len 0, sigclass 0x00
digest algo 8, begin of digest 27 28
hashed subpkt 2 len 4 (sig created 2015-06-29)
subpkt 16 len 8 (issuer key ID E149B3889E4DA08C)
data: [2047 bits]
```

- ▶ Signature parameters
  - ▶ sigclass: type of data (text, binary, User ID, etc.)
  - ▶ digest: Hash algorithm
  - ▶ pubkey: public-key algorithm
  - ▶ keyid: public key id
  - ▶ last: last onepass\_sig packet before the data?
- ▶ Parameters precede the data; no buffering required

# A Signed Message

```
$ echo -n foo | gpg2 -s | gpg2 --list-packets
# off=0 ctb=a3 tag=8 hlen=1 plen=0 indeterminate
:compressed packet: algo=1
# off=2 ctb=90 tag=4 hlen=2 plen=13
:onepass_sig packet: keyid E149B3889E4DA08C
version 3, sigclass 0x00, digest 8, pubkey 1, last=1
# off=17 ctb=cb tag=11 hlen=2 plen=9 new-ctb
:literal data packet:
mode b (62), created 1435588610, name="",
raw data: 3 bytes
# off=28 ctb=89 tag=2 hlen=3 plen=284
:signature packet: algo 1, keyid E149B3889E4DA08C
version 4, created 1435588610, md5len 0, sigclass 0x00
digest algo 8, begin of digest 27 28
hashed subpkt 2 len 4 (sig created 2015-06-29)
subpkt 16 len 8 (issuer key ID E149B3889E4DA08C)
data: [2047 bits]
```

- ▶ Actual data to sign
- ▶ Parameters
  - ▶ mode: binary, text or UTF-8
  - ▶ created: file's last modification date
  - ▶ name: filename

# A Signed Message

```
$ echo -n foo | gpg2 -s | gpg2 --list-packets
# off=0 ctb=a3 tag=8 hlen=1 plen=0 indeterminate
:compressed packet: algo=1
# off=2 ctb=90 tag=4 hlen=2 plen=13
:onepass_sig packet: keyid E149B3889E4DA08C
version 3, sigclass 0x00, digest 8, pubkey 1, last=1
# off=17 ctb=cb tag=11 hlen=2 plen=9 new-ctb
:literal data packet:
mode b (62), created 1435588610, name="",
raw data: 3 bytes
# off=28 ctb=89 tag=2 hlen=3 plen=284
:signature packet: algo 1, keyid E149B3889E4DA08C
version 4, created 1435588610, md5len 0, sigclass 0x00
digest algo 8, begin of digest 27 28
hashed subpkt 2 len 4 (sig created 2015-06-29)
subpkt 16 len 8 (issuer key ID E149B3889E4DA08C)
data: [2047 bits]
```

- ▶ The actual signature
  - ▶ Repetition of parameters from onepass\_sig packet
  - ▶ The hash
  - ▶ Additional subpackets (included in the hash)
    - ▶ Creation time
    - ▶ Full public key

# Public Keys

```
$ gpg2 --export testing | gpg2 --list-packets
# off=0 ctb=99 tag=6 hlen=3 plen=269
:public key packet:
  version 4, algo 1, created 1431979963, expires 0
  keyid: E149B3889E4DA08C
# off=272 ctb=b4 tag=13 hlen=2 plen=7
:user ID packet: "Testing"
# off=281 ctb=89 tag=2 hlen=3 plen=319
:signature packet: algo 1, keyid E149B3889E4DA08C
  version 4, created 1431979963, md5len 0, sigclass 0x13
  digest algo 8, begin of digest 7b 58
  hashed subpkt 2 len 4 (sig created 2015-05-18)
  hashed subpkt 27 len 1 (key flags: 03)
  hashed subpkt 9 len 4 (key expires after 100d0h0m)
  hashed subpkt 11 len 6 (pref-sym-algos: 9 8 7 3 2 1)...
# off=603 ctb=b9 tag=14 hlen=3 plen=269
:public sub key packet:
  version 4, algo 1, created 1431979963, expires 0
  keyid: AE19DAC58E678210
# off=875 ctb=89 tag=2 hlen=3 plen=293
:signature packet: algo 1, keyid E149B3889E4DA08C
...
```

- ▶ Packet format also used for serializing keys
- ▶ Includes preferences and supported features
  - ▶ Upload your keys regularly!

# Public Keys

```
$ gpg2 --export testing | gpg2 --list-packets
# off=0 ctb=99 tag=6 hlen=3 plen=269
:public key packet:
  version 4, algo 1, created 1431979963, expires 0
  keyid: E149B3889E4DA08C
# off=272 ctb=b4 tag=13 hlen=2 plen=7
:user ID packet: "Testing"
# off=281 ctb=89 tag=2 hlen=3 plen=319
:signature packet: algo 1, keyid E149B3889E4DA08C
  version 4, created 1431979963, md5len 0, sigclass 0x13
  digest algo 8, begin of digest 7b 58
  hashed subpkt 2 len 4 (sig created 2015-05-18)
  hashed subpkt 27 len 1 (key flags: 03)
  hashed subpkt 9 len 4 (key expires after 100d0h0m)
  hashed subpkt 11 len 6 (pref-sym-algos: 9 8 7 3 2 1)...
# off=603 ctb=b9 tag=14 hlen=3 plen=269
:public sub key packet:
  version 4, algo 1, created 1431979963, expires 0
  keyid: AE19DAC58E678210
# off=875 ctb=89 tag=2 hlen=3 plen=293
:signature packet: algo 1, keyid E149B3889E4DA08C
...
```

## ► Public key / subkey

# Public Keys

```
$ gpg2 --export testing | gpg2 --list-packets
# off=0 ctb=99 tag=6 hlen=3 plen=269
:public key packet:
  version 4, algo 1, created 1431979963, expires 0
  keyid: E149B3889E4DA08C
# off=272 ctb=b4 tag=13 hlen=2 plen=7
:user ID packet: "Testing"
# off=281 ctb=89 tag=2 hlen=3 plen=319
:signature packet: algo 1, keyid E149B3889E4DA08C
  version 4, created 1431979963, md5len 0, sigclass 0x13
  digest algo 8, begin of digest 7b 58
  hashed subpkt 2 len 4 (sig created 2015-05-18)
  hashed subpkt 27 len 1 (key flags: 03)
  hashed subpkt 9 len 4 (key expires after 100d0h0m)
  hashed subpkt 11 len 6 (pref-sym-algos: 9 8 7 3 2 1)...
# off=603 ctb=b9 tag=14 hlen=3 plen=269
:public sub key packet:
  version 4, algo 1, created 1431979963, expires 0
  keyid: AE19DAC58E678210
# off=875 ctb=89 tag=2 hlen=3 plen=293
:signature packet: algo 1, keyid E149B3889E4DA08C
...
```

## ► Self-signature

- Always uses primary key
- Links subkey to primary

# Public Keys

```
$ gpg2 --export testing | gpg2 --list-packets
# off=0 ctb=99 tag=6 hlen=3 plen=269
:public key packet:
  version 4, algo 1, created 1431979963, expires 0
  keyid: E149B3889E4DA08C
# off=272 ctb=b4 tag=13 hlen=2 plen=7
:user ID packet: "Testing"
# off=281 ctb=89 tag=2 hlen=3 plen=319
:signature packet: algo 1, keyid E149B3889E4DA08C
  version 4, created 1431979963, md5len 0, sigclass 0x13
  digest algo 8, begin of digest 7b 58
  hashed subpkt 2 len 4 (sig created 2015-05-18)
  hashed subpkt 27 len 1 (key flags: 03)
  hashed subpkt 9 len 4 (key expires after 100d0h0m)
  hashed subpkt 11 len 6 (pref-sym-algos: 9 8 7 3 2 1)...
# off=603 ctb=b9 tag=14 hlen=3 plen=269
:public sub key packet:
  version 4, algo 1, created 1431979963, expires 0
  keyid: AE19DAC58E678210
# off=875 ctb=89 tag=2 hlen=3 plen=293
:signature packet: algo 1, keyid E149B3889E4DA08C
...
```

- ▶ Signature data
  - ▶ Key's properties
  - ▶ User preference
  - ▶ Supported features

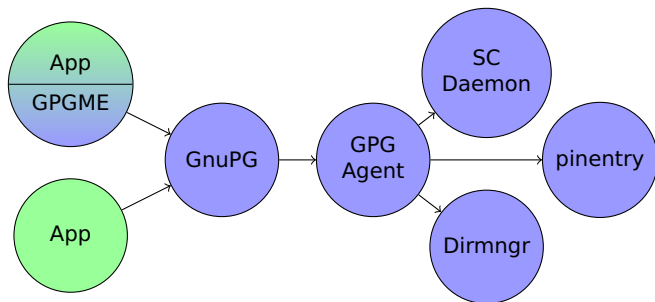
OpenPGP

GnuPG's Architecture

Good Practices

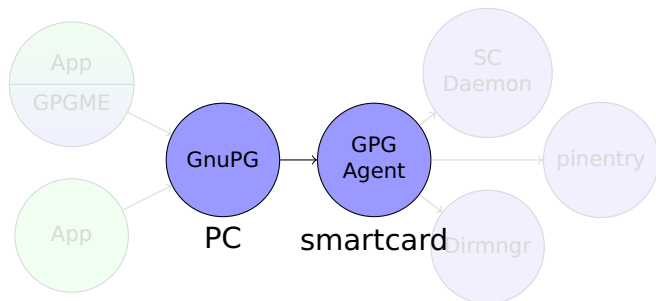
Neat Tricks

# GnuPG's Architecture



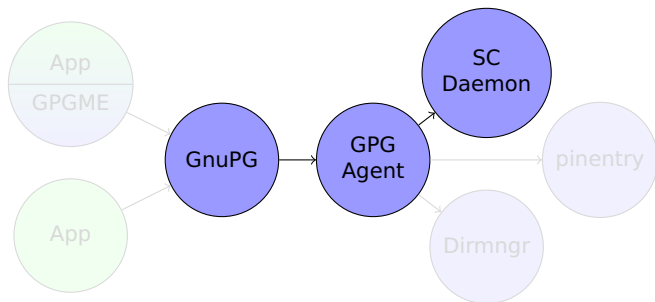
- ▶ Multi-server architecture
  - ▶ GPG is not a library!
    - ▶ GPGME *is* a library
    - ▶ Provides convenient APIs
    - ▶ Communicates with GPG
  - ▶ Components in their own address spaces
  - ▶ Reduces impact of bugs

# GnuPG's Architecture



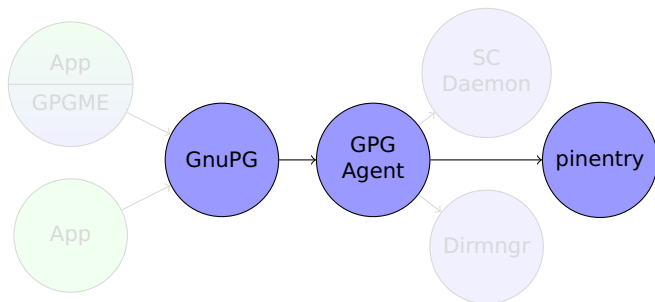
- ▶ GPG: Low security
  - ▶ Session encryption
  - ▶ Encoding, etc.
- ▶ GPG Agent: High security
  - ▶ Manages private key and passwords
  - ▶ Delegates to servers
- ▶ Separation similar to that of a PC and smartcard

# GnuPG's Architecture



- ▶ Smartcard Daemon
  - ▶ Interacts with smartcards (directly or via PC/SC)
  - ▶ Typically packaged separately as `scdaemon`

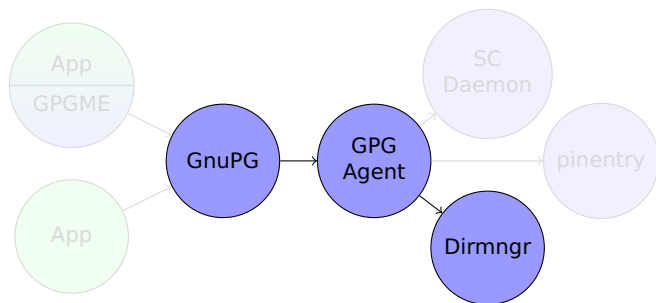
# GnuPG's Architecture



## ► Pinentry

- For user interactions
  - Request passphrase
  - Ask questions
- Multiple implementations
  - Tighter integration
  - Different security properties

# GnuPG's Architecture

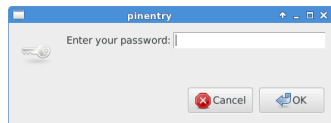


- ▶ Directory manager
  - ▶ Interacts with keyserver (HKP, ldap, http)
    - ▶ `gpg2 --search-keys email@example.org`
    - ▶ `gpg2 --recv-key keyid`
    - ▶ etc.
  - ▶ Certificate and CRL cache

# Assuan

- ▶ Components communicate using Assuan protocol
  - ▶ IPC protocol
  - ▶ Pipe / socket based
  - ▶ Very simple, text-based interface
  - ▶ No interface definition language (IDL)
  - ▶ Example:

```
$ pinentry
OK Your orders please
setprompt Enter your password:
OK
getpin
D 123abc
OK
```



- ▶ Use `gpg-connect-agent` to connect to the running GPG Agent or `dirmngr`

# watchgnupg

- ▶ Tool for gathering log entries
  - ▶ In `gpg-agent.conf`, add:
    - ▶ `log-file socket:///home/USER/.gnupg/S.log`
    - ▶ `debug-level basic # (or advanced or expert)`
  - ▶ Run:  
`$ watchgnupg --force /home/USER/.gnupg/S.log`

OpenPGP

GnuPG's Architecture

Good Practices

Neat Tricks

# Private Key Management

## ▶ Online

- ▶ gpg2 --gen-key
- ▶ Key stored locally
- ▶ **Low security**: must trust all local software

## ▶ Offline

- ▶ Key stored on a smartcard (GnuK, Yubikey Neo, etc.)
- ▶ Should use subkeys
  - ▶ Setup slightly more complicated
- ▶ Should store backups on a USB stick
  - ▶ Can't export private key from smartcard
- ▶ **Much higher security**
  - ▶ Crypto can only be when key is inserted
  - ▶ **But**, often not obvious what the operation is

- ▶ Note: easier to explain crypto when using a smartcard

# Private Key Management

- ▶ Online
  - ▶ gpg2 --gen-key
  - ▶ Key stored locally
  - ▶ **Low security**: must trust all local software
- ▶ Offline
  - ▶ Key stored on a smartcard (GnuK, Yubikey Neo, etc.)
  - ▶ Should use subkeys
    - ▶ Setup slightly more complicated
  - ▶ Should store backups on a USB stick
    - ▶ Can't export private key from smartcard
  - ▶ **Much higher security**
    - ▶ Crypto can only be when key is inserted
    - ▶ **But**, often not obvious what the operation is
- ▶ Note: easier to explain crypto when using a smartcard

# Private Key Management

- ▶ Online
  - ▶ gpg2 --gen-key
  - ▶ Key stored locally
  - ▶ **Low security**: must trust all local software
- ▶ Offline
  - ▶ Key stored on a smartcard (GnuK, Yubikey Neo, etc.)
  - ▶ Should use subkeys
    - ▶ Setup slightly more complicated
  - ▶ Should store backups on a USB stick
    - ▶ Can't export private key from smartcard
  - ▶ **Much higher security**
    - ▶ Crypto can only be when key is inserted
    - ▶ **But**, often not obvious what the operation is
- ▶ Note: easier to explain crypto when using a smartcard

# Offline Keys

- ▶ Use Tails!!!
  - ▶ Hardened
  - ▶ Wipes memory on shutdown
- ▶ Managing the key:
  - ▶ Boot from a USB stick
    - ▶ **Medium Security**
    - ▶ BIOS might be infected, etc.
  - ▶ Use a dedicated offline computer
    - ▶ Old IBM x40 or x60 costs <50 Euros on ebay
    - ▶ Remove wireless network card!
    - ▶ **High security**
    - ▶ **But**, still susceptible to Bad USB!

# Generating a *Secure* Passphrase

- ▶ Generating a secure passphrase is hard
  - ▶ “Assume your adversary is capable of one trillion guesses per second.” -Snowden
  - ▶ To withstand one year, need 65 bits of entropy!
  - ▶ How to measure a password's entropy?
  - ▶ Need a random password
  - ▶ But that's impossible to memorize
  - ▶ Unless we encode it smartly!

# Generating a *Secure* Passphrase

- ▶ Generating a secure passphrase is hard
  - ▶ “Assume your adversary is capable of one trillion guesses per second.” -Snowden
  - ▶ To withstand one year, need 65 bits of entropy!
  - ▶ How to measure a password's entropy?
  - ▶ Need a random password
  - ▶ But that's impossible to memorize
  - ▶ Unless we encode it smartly!

# Diceware

- ▶ Encode using a simple word list
  - ▶ /dev/random? 1k words (10-bits entropy per word)
  - ▶ dice?  $6^4 = 1296$  words (10.3-bits entropy)
- ▶ Secure even if adversary knows the word list!
- ▶ Examples:
  - ▶ 1. able
  - ▶ 2. about
  - ▶ 3. above
  - ▶ ...
- ▶ Required length:
  - ▶ 80 bits = good = 8 words
  - ▶ 120 bits = strong = 12 words
- ▶ Examples:
  - ▶ percent burst identify smash opposite ready blind stab
  - ▶ pipe after harm person split seize radar about

# Diceware

- ▶ Encode using a simple word list
  - ▶ /dev/random? 1k words (10-bits entropy per word)
  - ▶ dice?  $6^4 = 1296$  words (10.3-bits entropy)
- ▶ Secure even if adversary knows the word list!
- ▶ Examples:
  - ▶ 1. able
  - ▶ 2. about
  - ▶ 3. above
  - ▶ ...
- ▶ Required length:
  - ▶ 80 bits = good = 8 words
  - ▶ 120 bits = strong = 12 words
- ▶ Examples:
  - ▶ percent burst identify smash opposite ready blind stab
  - ▶ pipe after harm person split seize radar about

# Word Lists

- ▶ Diceware (8k)
- ▶ PGP Biometric word list (512)
- ▶ Voice of America's simple English word list (1.5k)

# Avoiding Man in the Middle Attacks

- ▶ Key signing parties are for geeks
- ▶ Exchanging fingerprints in person is inconvenient
- ▶ Use the telephone!

```
$ gpg2 --recv-key 630052D9
```

```
$ gpg2 --with-icao-spelling --fingerprint 630052D9
```

```
pub  rsa3744/630052D9 2015-04-07 [expires: 2025-04-04]
```

```
Key fingerprint = 8F17 7771 18A3 3DDA 9BA4 8E62 AACB 3243 0
```

```
"Eight Foxtrot One Seven Seven Seven Sev
```

```
...
```

```
$ gpg2 --sign-key 630052D9 # or --lsign-key
```

- ▶ Secure enough for all but the most paranoid
- ▶ Much more secure than no check

# Key Management

- ▶ When you get a signed message, fetch the key
- ▶ Refresh keys regularly
  - ▶ Why?
    - ▶ New preferences
    - ▶ Revocation certificates
  - ▶ How?
    - ▶ Don't use `gpg2 --refresh-keys`
    - ▶ Install `parcimonie`
    - ▶ Uses `tor`
    - ▶ Random intervals between each key refresh

# Key Disclosure

- ▶ You have to disclose the encryption key for a message?
- ▶ **Don't disclose your private key!**
- ▶ This allows decryption of all messages
- ▶ Just disclose the session key.

```
$ echo | gpg2 -e -r keyid | gpg2 --show-session-key  
...  
gpg: session key: '9:576EE31...'
```

# Don't backup the RNG's seed!

- ▶ Exclude `.gnupg/random_seek` from backups!

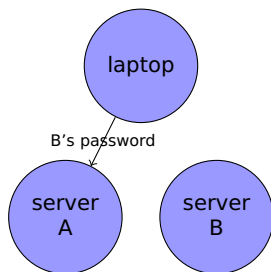
OpenPGP

GnuPG's Architecture

Good Practices

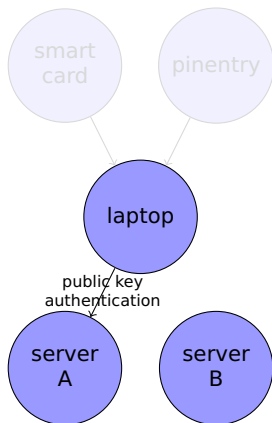
Neat Tricks

# ssh: Keys Instead of Passwords



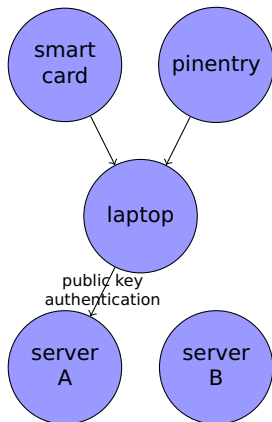
- ▶ Using keys means password is not sent to server
  - ▶ Ever enter password for a different server?
  - ▶ You've just disclosed your password!

# ssh keys



- ▶ OpenSSH stores private keys on hard drive
- ▶ Keys are protected by a passphrase
- ▶ Passphrase is cached by ssh agent

# ssh keys



- ▶ GnuPG implements the ssh agent protocol
- ▶ GnuPG can use keys stored on a smart card

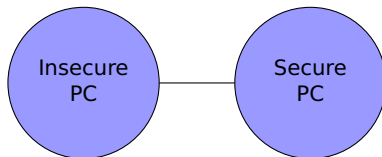
# GnuPG's ssh agent: configuration

- ▶ Add enable-ssh-support to  
  .gnupg/gpg-agent.conf
- ▶ Restart gpg agent
- ▶ Add public key to .ssh/authorized\_keys file
- ▶ public key obtained by doing:  
  \$ ssh-add -L  
  ssh-rsa AAAAB3NzaC1...zyt cardno:000603016636

# Remote gpg-agent

- ▶ gpg can use a remote gpg-agent
  - ▶ Running on another computer
  - ▶ Running as a different user

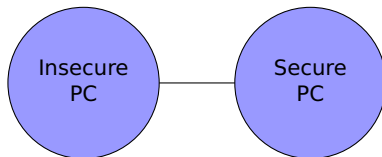
# How it works



- ▶ Create a new user, `gpg`
- ▶ On secure pc, add the following to `.gnupg/gpg-agent.conf`:  
`extra-socket /home/gpg/.gnupg/S.gpg-agent-remote`
- ▶ On insecure pc, run the following to forward the port:  

```
$ ssh -f -o ExitOnForwardFailure=yes -o StreamLocalBindUnlink=yes \  
> -L /home/neal/.gnupg/S.gpg-agent:/home/gpg/.gnupg/S.gpg-agent-remote \  
> gpg@localhost bash -c 'while sleep 5; do echo NOP; done | gpg-connect-agent'
```
- ▶ Requires OpenSSH  $\geq 6.7$  (Unix Domain Sockets)

# How it works

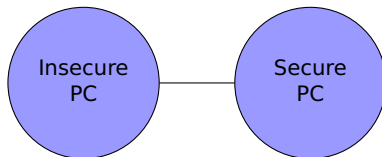


- ▶ Create a new user, `gpg`
- ▶ On secure pc, add the following to `.gnupg/gpg-agent.conf`:  
`extra-socket /home/gpg/.gnupg/S.gpg-agent-remote`
- ▶ On insecure pc, run the following to forward the port:  

```
$ ssh -f -o ExitOnForwardFailure=yes -o StreamLocalBindUnlink=yes \  
> -L /home/neal/.gnupg/S.gpg-agent:/home/gpg/.gnupg/S.gpg-agent-remote \  
> gpg@localhost bash -c 'while sleep 5; do echo NOP; done | gpg-connect-agent'
```

  - ▶ If forwarding fails, exit
  - ▶ If the socket to be forwarded already exists, remove it first

# How it works

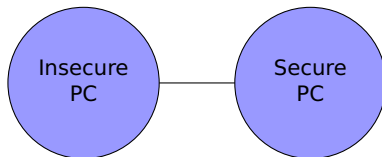


- ▶ Create a new user, `gpg`
- ▶ On secure pc, add the following to `.gnupg/gpg-agent.conf`:  
`extra-socket /home/gpg/.gnupg/S.gpg-agent-remote`
- ▶ On insecure pc, run the following to forward the port:  

```
$ ssh -f -o ExitOnForwardFailure=yes -o StreamLocalBindUnlink=yes \  
> -L /home/neal/.gnupg/S.gpg-agent:/home/gpg/.gnupg/S.gpg-agent-remote \  
> gpg@localhost bash -c 'while sleep 5; do echo NOP; done | gpg-connect-agent'
```

  - ▶ Forwards the file `.../S.gpg-agent` on insecure
  - ▶ To the file `.../S.gpg-agent-remote` on secure
  - ▶ `ssh` won't expand tildes

# How it works



- ▶ Create a new user, `gpg`
- ▶ On secure pc, add the following to `.gnupg/gpg-agent.conf`:  
`extra-socket /home/gpg/.gnupg/S.gpg-agent-remote`
- ▶ On insecure pc, run the following to forward the port:  

```
$ ssh -f -o ExitOnForwardFailure=yes -o StreamLocalBindUnlink=yes \  
> -L /home/neal/.gnupg/S.gpg-agent:/home/gpg/.gnupg/S.gpg-agent-remote \  
> gpg@localhost bash -c 'while sleep 5; do echo NOP; done | gpg-connect-agent'
```

  - ▶ Loop keeps connection opened and port forwarded
  - ▶ Exits when `gpg-agent` exits

# Thanks!

- ▶ Slides will be online at [www.gnupg.org](http://www.gnupg.org)
- ▶ More resources:
  - ▶ Riseup: <https://help.riseup.net/en/security/message-security/openpgp/best-practices>
  - ▶ The grugq (for the truly paranoid): <https://gist.github.com/grugq/03167bed45e774551155>
- ▶ If you like our work, consider donating!
  - ▶ <https://www.gnupg.org/donate>

# Copyright

This presentation is Copyright 2015, by Neal H. Walfield. License: CC BY-SA 2.0.