bootc's wiki

Release 1.0

Chris Boot & contributors

Jun 02, 2017

Contents

This is a wiki of sorts where I have documented various things that I've worked on or figured out. Hopefully you'll find useful things here.

Contributions are welcome, but this "wiki" doesn't have a pretty editor. Instead it uses Sphinx to build the pages from a Git repository hosted on GitHub.

CHAPTER 1

Table of Contents

GnuPG

Generating Keys

Note: this documentation has mostly just been copied from my blog and still needs some attention and updating.

Getting started and requirements

I'm going to assume you have already checked your computer is secure, so that nobody can eavesdrop in the key creation process and/or access your backup medium while it is unlocked. Covering this requirement is a big topic and outside the scope of this guide.

This guide also assumes you are using Debian GNU/Linux 7.0 (wheezy), which includes all of the required software in its archive. The cards I am using are the g10-code OpenPGP cards housed in Gemalto IDBridge K30 (USB Shell Token V2) readers. A colleague has tested the IDBridge K50 / Shell Token V3 readers and they work fine as well; the ACS ACR83 full-size smart card PIN pad reader does *not* work because it doesn't support the extended-length commands required by the OpenPGP cards. Despite all of the documentation, the cards *do* support 4096-bit keys in all three slots simultaneously, e.g. you can store 3 × 4096-bit keys on one OpenPGP smart card.

The packages required for this guide include `gnupg2 gnupg-agent pcscd pcsc-tools libccid scdaemon`. These are:

- `gnupg2`: The 2.x branch of GnuPG. This version is required for key lengths greater than 2048 bits on the smart cards. This requires `gnupg-agent` and `scdaemon` for proper operation together with smart cards.
- `pcscd` and `pcsc-tools`: The PC/SC smart card software and its utilities.
- `libccid`: Driver library for USB CCID smart card readers.

Note that gnupg2 2.0.19 (the wheezy version) contains a bug that means data decryption using a key stored on a smart card fails. This isn't important for this guide, but is likely to cause you issues in daily use. You will want to use 2.0.20 or better which is in jessie (Debian testing); I have re-compiled this version and it is available in my own apt repository.

Setting up PC/SC and testing your reader

Make sure all of the required packages are installed first and that `pcscd` is running. Next, with your card reader unplugged, run `pcsc_scan` as `root`. It should show a few lines of output and end on a line reading "Waiting for the first reader...". Now plug in your smart card reader and smart card; you should see lots of output resulting from detecting and probing your card and finally end up with a few lines that say "Possibly identified card [...] GnuPG card V2".

If this is *not* the case, have a look in `/var/log/syslog` and/or restart `pcscd` in case the driver for your reader wasn't loaded correctly.

Next, we must enable our non-root user to do the same. To do this, ensure your user is in the `pcscd` group by running the following command as `root`:

```
usermod -a -G pcscd «user»
```

You must then logout of your session completely and log back in for the change to take effect. You should be able to see the group using the `id` command, for example:

```
bootc@ripley ~ $ id -Gn
bootc disk [...] libvirt pcscd
```

Now, run `pcsc_scan` again but using your own user this time, not root. The results should be the same as when you ran it as root.

Testing the smart card with GnuPG and setting up the basics

Now that you have the reader and card working with PC/SC, we need to check that GnuPG can use it. We'll use the `gpg2 --card-status` command to read the empty card, which should show something along the lines of the following:

```
bootc@ripley ~ $ gpg2 --card-status
Application ID ...: D2760001240102000005000001BD40000
Version ..........: 2.0
Manufacturer .....: ZeitControl
Serial number ....: 00001BD4
Name of cardholder: [not set]
Language prefs ...: de
Sex ..............: unspecified
URL of public key : [not set]
Login data .......: [not set]
Signature PIN ....: forced
Key attributes ...: 2048R 2048R 2048R
Max. PIN lengths .: 32 32 32
PIN retry counter : 3 0 3
Signature counter : 0
Signature key ....: [none]
Encryption key....: [none]
Authentication key: [none]
General key info..: [none]
```

Congratulations, you now have a working OpenPGP smart card setup. Let's now change a few mundane details like the cardholder name, language preference and sex. Note that none of these fields are used by GnuPG, so you may choose to leave them set as they are if you wish. To change these values, use `gpg2 --card-edit`:

```
bootc@ripley ~ $ gpg2 --card-edit

```

```
Application ID ...: D2760001240102000005000001BD40000
Version ..........: 2.0
Manufacturer .....: ZeitControl
Serial number ....: 00001BD4
Name of cardholder: [not set]
Language prefs ...: de
Sex ..............: unspecified
URL of public key : [not set]
Login data .......: [not set]
Signature PIN ....: forced
Key attributes ...: 2048R 2048R 2048R
Max. PIN lengths .: 32 32 32
PIN retry counter : 3 0 3
Signature counter : 0
Signature key ....: [none]
Encryption key....: [none]
Authentication key: [none]
General key info..: [none]

gpg/card> admin
Admin commands are allowed

gpg/card> name
Cardholder's surname: Boot
Cardholder's given name: Chris

gpg/card>
```

The default "User PIN" code on the cards is `123456`, and the "Admin PIN" is `12345678`. I'll leave it as an exercise to the reader to change the language (using the `lang` command) and sex (using the `sex` command).

Changing the PIN codes

Now is probably a good time to ensure the PIN codes on the card are changed to sensible values. Note that despite these being called PIN codes, the cards will accept full ASCII passwords, and not just numbers. You may, however, want to restrict yourself to just numbers if you want to use a pin-pad reader for Secure PIN Entry, as they usually only have numeric buttons on them.

There are three different PIN codes on an OpenPGP 2.0 card. The User PIN, which is used for unlocking of the keys for signing, decryption and authentication, the Admin PIN, used for setting up the card including changing the cardholder name and similar details, and finally a Resetting Code, which can be used to unblock the User PIN instead of using the Admin PIN (the Resetting Code is disabled by default).

You can change all 3 of the PIN codes using either the `passwd` command within `gpg2 --card-edit`, or using `gpg2 --change-pin`.

Creating a secure container for backups

Your private keys need to be backed up somewhere safe, or you'll be sorry if you lose your keys and have to go through the entire process of creating a new key and having it signed by all your acquaintances. That being said, the backups themselves need to be kept securely so that it cannot be tampered with.

Most GnuPG guides will err on the side of extreme paranoia and insist that you create your keys while offline (e.g. disconnected from the internet) and then keep your backups both encrypted and locked away in a safe. That may be good advice, but it's too restrictive for me.

Instead, I'll opt to keep my backups in an encrypted file system container using LUKS, and sign the file with my new GPG key so that any tampering should show up. I'll then back up both the encrypted filesystem and its signature remotely. Yes, remotely, on the Internet. I'll rely on the two different passwords on the LUKS container and then within GnuPG itself to keep my keys secure.

Without further ado, here is how I set up my LUKS container, and pointed GnuPG at it for key creation. First, create a ~10MB file to act as the container. I used `/dev/random` as the source of random bytes for this because I have a Simtec Entropy Key and have plenty of real random data around; you likely want to use `/dev/urandom` instead if you don't have such a thing, otherwise you'll be waiting a *very* long time for this to complete:

```
bootc@ripley ~ $ dd if=/dev/urandom of=gpg-key-backup.luks.img iflag=fullblock bs=1M␣
↪count=10
10+0 records in
10+0 records out
10485760 bytes (10 MB) copied, 0.56882 s, 18.4 MB/s
```

Now, create the LUKS container with `cryptsetup`:

```
bootc@ripley ~ $ sudo -s
[sudo] password for bootc:
ripley bootc # chmod go= gpg-key-backup.luks.img
ripley bootc # cryptsetup --verbose --verify-passphrase luksFormat gpg-key-backup.
↪luks.img

WARNING!
========
This will overwrite data on gpg-key-backup.luks.img irrevocably.

Are you sure? (Type uppercase yes): YES
Enter LUKS passphrase:
Verify passphrase:
Command successful.
ripley bootc #
```

Now you need to open the encrypted container, put a filesystem on it, and mount it so that we can use it:

```
ripley bootc # cryptsetup --verbose luksOpen gpg-key-backup.luks.img gpg-key-backup
Enter passphrase for /home/bootc/gpg-key-backup.luks.img:
Key slot 0 unlocked.
Command successful.
ripley bootc # mkfs.ext4 /dev/mapper/gpg-key-backup
mke2fs 1.42.5 (29-Jul-2012)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
Stride=0 blocks, Stripe width=0 blocks
2048 inodes, 8192 blocks
409 blocks (4.99%) reserved for the super user
First data block=1
Maximum filesystem blocks=8388608
1 block group
8192 blocks per group, 8192 fragments per group
2048 inodes per group

Allocating group tables: done
Writing inode tables: done
Creating journal (1024 blocks): done
```

```
Writing superblocks and filesystem accounting information: done

ripley bootc # mkdir /mnt/gpg-key-backup
ripley bootc # mount -t ext4 -o journal_checksum /dev/mapper/gpg-key-backup /mnt/gpg-
↪key-backup
ripley bootc # chown bootc: /mnt/gpg-key-backup
ripley bootc # chmod go= /mnt/gpg-key-backup
ripley bootc # exit
bootc@ripley ~ $
```

We can now carry on as non-root again. Finally, we need to create a directory for GnuPG to use as its 'home' directory, and set an environment variable to point to it. Normally, GnuPG uses `~/.gnupg` for this purpose, but we want to make sure the keys go straight into our secure area.

```
bootc@ripley ~ $ mkdir /mnt/gpg-key-backup/gnupghome
bootc@ripley ~ $ chmod go= /mnt/gpg-key-backup/gnupghome
bootc@ripley ~ $ export GNUPGHOME=/mnt/gpg-key-backup/gnupghome
bootc@ripley ~ $
```

Creating your primary key

This is the big moment! We're now creating the key that will be our identity:

```
bootc@ripley ~ $ gpg2 --gen-key
gpg (GnuPG) 2.0.20; Copyright (C) 2013 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection? 4 [ yes, sign-only is correct, we'll use sub-keys for encryption ]
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 4096 [ no reason to go any smaller with modern␣
↪hardware ]
Requested keysize is 4096 bits
Please specify how long the key should be valid.
         0 = key does not expire
        = key expires in n days
      w = key expires in n weeks
      m = key expires in n months
      y = key expires in n years
Key is valid for? (0)
Key does not expire at all
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: Chris Boot
Email address: bootc@bootc.net
Comment: TESTING ONLY [ you most likely want to leave this blank ]
You selected this USER-ID:
    "Chris Boot (TESTING ONLY)"
```

```
Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
You need a Passphrase to protect your secret key.
[ pinentry will ask you for a password, use a secure one different from the LUKS␣
↪passphrase ]

We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: key 75FFB60A marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
pub   4096R/75FFB60A 2013-06-07
      Key fingerprint = 94B1 3318 3E46 0315 6730  0AC1 5F0C DB26 75FF B60A
uid                  Chris Boot (TESTING ONLY)

Note that this key cannot be used for encryption.  You may want to use
the command "--edit-key" to generate a subkey for this purpose.
```

Adding identities

You may want to use your key for multiple email addresses for example. You should add the identities you plan to use as early as possible, as when your key is signed by other people it's actually each identity that is signed, rather than the key as a whole. This means you can't create a "Joe Bloggs" identity, then replace it with a "Bill Gates" identity and retain all the signatures.

[note at this point I created a new smaller key for the purposes of this tutorial, so the key IDs are different to what was shown earlier]

This is how you add a new identity:

```
bootc@ripley ~ $ gpg2 --edit-key 27703CF0
gpg (GnuPG) 2.0.20; Copyright (C) 2013 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Secret key is available.

pub  1024R/27703CF0  created: 2013-06-08  expires: never       usage: SC
                     trust: ultimate      validity: ultimate
[ultimate] (1). Chris Boot (TESTING ONLY) <bootc@bootc.net>

gpg> adduid
Real name: Chris Boot
Email address: otheremail@bootc.net
Comment: STILL TESTING
You selected this USER-ID:
    "Chris Boot (STILL TESTING) <otheremail@bootc.net>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O

You need a passphrase to unlock the secret key for
user: "Chris Boot (TESTING ONLY) <bootc@bootc.net>"
1024-bit RSA key, ID 27703CF0, created 2013-06-08
```

```
pub  1024R/27703CF0  created: 2013-06-08  expires: never       usage: SC
                     trust: ultimate      validity: ultimate
[ultimate] (1)  Chris Boot (TESTING ONLY) <bootc@bootc.net>
[ unknown] (2). Chris Boot (STILL TESTING) <otheremail@bootc.net>

gpg> save
bootc@ripley ~ $
```

Just keep running `adduid` for each identity you want to add. Finally, you want to set the primary UID on your key; this is the main one that shows up when there isn't space to list all of the identities, but it has no other special meaning - it's purely cosmetic.

```
bootc@ripley ~ $ gpg2 --edit-key 27703CF0
gpg (GnuPG) 2.0.20; Copyright (C) 2013 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Secret key is available.

pub  1024R/27703CF0  created: 2013-06-08  expires: never       usage: SC
                     trust: ultimate      validity: ultimate
[ultimate] (1). Chris Boot (STILL TESTING) <otheremail@bootc.net>
[ultimate] (2)  Chris Boot (TESTING ONLY) <bootc@bootc.net>

gpg> uid 2

pub  1024R/27703CF0  created: 2013-06-08  expires: never       usage: SC
                     trust: ultimate      validity: ultimate
[ultimate] (1). Chris Boot (STILL TESTING) <otheremail@bootc.net>
[ultimate] (2)* Chris Boot (TESTING ONLY) <bootc@bootc.net>

gpg> primary

You need a passphrase to unlock the secret key for
user: "Chris Boot (STILL TESTING) <otheremail@bootc.net>"
1024-bit RSA key, ID 27703CF0, created 2013-06-08

pub  1024R/27703CF0  created: 2013-06-08  expires: never       usage: SC
                     trust: ultimate      validity: ultimate
[ultimate] (1)  Chris Boot (STILL TESTING) <otheremail@bootc.net>
[ultimate] (2)* Chris Boot (TESTING ONLY) <bootc@bootc.net>

gpg> save
bootc@ripley ~ $
```

The primary key is denoted by the `.` next to the UID number, the `*` is the identity selected using the `uid` command.

Creating your sub-keys

As discussed before, we'll use the primary key only for signing other people's keys and our own identities - not for daily signing or encryption use. We'll now create sub-keys for signing, encryption and authentication. Note that the `--expert` argument to gpg2 is required to create the correct keys for use with the OpenPGP smart card.

```
bootc@ripley ~ $ gpg2 --expert --edit-key 27703CF0
gpg (GnuPG) 2.0.20; Copyright (C) 2013 Free Software Foundation, Inc.
```

```
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Secret key is available.

pub 1024R/27703CF0 created: 2013-06-08 expires: never usage: SC
                     trust: ultimate validity: ultimate
[ultimate] (1). Chris Boot (TESTING ONLY) <bootc@bootc.net>
[ultimate] (2)  Chris Boot (STILL TESTING) <otheremail@bootc.net>

gpg> addkey
Key is protected.

You need a passphrase to unlock the secret key for
user: "Chris Boot (TESTING ONLY) <bootc@bootc.net>"
1024-bit RSA key, ID 27703CF0, created 2013-06-08

Please select what kind of key you want:
(3) DSA (sign only)
(4) RSA (sign only)
(5) Elgamal (encrypt only)
(6) RSA (encrypt only)
(7) DSA (set your own capabilities)
(8) RSA (set your own capabilities)
Your selection? 8 [ it's very important to select this option ]

Possible actions for a RSA key: Sign Encrypt Authenticate
Current allowed actions: Sign Encrypt

(S) Toggle the sign capability
(E) Toggle the encrypt capability
(A) Toggle the authenticate capability
(Q) Finished

Your selection? E [ take the encrypt action out of the key ]

Possible actions for a RSA key: Sign Encrypt Authenticate
Current allowed actions: Sign [ this is what you want to be left with ]

(S) Toggle the sign capability
(E) Toggle the encrypt capability
(A) Toggle the authenticate capability
(Q) Finished

Your selection? Q
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 1024 [ you want to use 4096 here ]
Requested keysize is 1024 bits
Please specify how long the key should be valid.
0 = key does not expire
<n> = key expires in n days
<n>w = key expires in n weeks
<n>m = key expires in n months
<n>y = key expires in n years
Key is valid for? (0) 1y
Key expires at Sun 08 Jun 2014 19:23:24 BST
Is this correct? (y/N) y
Really create? (y/N) y
```

```
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.

pub 1024R/27703CF0 created: 2013-06-08 expires: never usage: SC
                   trust: ultimate validity: ultimate
sub 1024R/41320871 created: 2013-06-08 expires: 2014-06-08 usage: S
[ultimate] (1). Chris Boot (TESTING ONLY) <bootc@bootc.net>
[ultimate] (2)  Chris Boot (STILL TESTING) <otheremail@bootc.net>

gpg>
```

Note that I created a 1024-bit key for this blog post, you should create a 4096-bit key instead. I also chose a 1-year validity on the subkey, and you should too. Subkey validity can be easily extended using your primary key, and is a useful safety net in case you lose your key.

You should now repeat the `addkey` command twice more, once to create an encryption-only key, and once to create an authentication-only key. Don't forget to `save` your key when you have finished.

You should end up with something that looks a bit like this:

```
pub 1024R/27703CF0 created: 2013-06-08 expires: never usage: SC
                   trust: ultimate validity: ultimate
sub 1024R/41320871 created: 2013-06-08 expires: 2014-06-08 usage: S
sub 1024R/B47AED2F created: 2013-06-08 expires: 2014-06-08 usage: E
sub 1024R/4495E34E created: 2013-06-08 expires: 2014-06-08 usage: A
[ultimate] (1). Chris Boot (TESTING ONLY) <bootc@bootc.net>
[ultimate] (2)  Chris Boot (STILL TESTING) <otheremail@bootc.net>
```

Moving keys to your smart cards

I chose to use two smart cards; if you only want to use one, skip moving the primary key and only move across the sub-keys. You can simply mount your LUKS-encrypted volume on the rare occasions you need to use your primary key.

First, we'll make a copy of the gnupghome directory we created earlier, and operate only on the copy. For some reason, keys can only be moved to the card, not copied - so we need to operate on a copy so that our backup is not wiped out:

```
bootc@ripley ~ $ rsync -a /mnt/gpg-key-backup/gnupghome/ /mnt/gpg-key-backup/
↪gnupghome-temp/
bootc@ripley ~ $ export GNUPGHOME=/mnt/gpg-key-backup/gnupghome-temp
bootc@ripley ~ $
```

Now, we can move our primary key to the first of our OpenPGP cards:

```
bootc@ripley ~ $ gpg2 --edit-key 27703CF0
gpg (GnuPG) 2.0.20; Copyright (C) 2013 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Secret key is available.

pub  1024R/27703CF0  created: 2013-06-08  expires: never       usage: SC
                     trust: ultimate      validity: ultimate
sub  1024R/41320871  created: 2013-06-08  expires: 2014-06-08  usage: S
sub  1024R/B47AED2F  created: 2013-06-08  expires: 2014-06-08  usage: E
```

```
sub  1024R/4495E34E  created: 2013-06-08  expires: 2014-06-08  usage: A
[ultimate] (1). Chris Boot (TESTING ONLY) <bootc@bootc.net>
[ultimate] (2)  Chris Boot (STILL TESTING) <otheremail@bootc.net>

gpg> toggle

sec  1024R/27703CF0  created: 2013-06-08  expires: never
ssb  1024R/41320871  created: 2013-06-08  expires: never
ssb  1024R/B47AED2F  created: 2013-06-08  expires: never
ssb  1024R/4495E34E  created: 2013-06-08  expires: never
(1)  Chris Boot (TESTING ONLY) <bootc@bootc.net>
(2)  Chris Boot (STILL TESTING) <otheremail@bootc.net>

gpg> keytocard
Really move the primary key? (y/N) y
Signature key ....: [none]
Encryption key....: [none]
Authentication key: [none]

Please select where to store the key:
   (1) Signature key
   (3) Authentication key
Your selection? 1

You need a passphrase to unlock the secret key for
user: "Chris Boot (TESTING ONLY) <bootc@bootc.net>"
1024-bit RSA key, ID 27703CF0, created 2013-06-08

sec  1024R/27703CF0  created: 2013-06-08  expires: never
                     card-no: 0005 00001BD4
ssb  1024R/41320871  created: 2013-06-08  expires: never
ssb  1024R/B47AED2F  created: 2013-06-08  expires: never
ssb  1024R/4495E34E  created: 2013-06-08  expires: never
(1)  Chris Boot (TESTING ONLY) <bootc@bootc.net>
(2)  Chris Boot (STILL TESTING) <otheremail@bootc.net>

gpg> save
bootc@ripley ~ $
```

And now for the sub-keys (insert the other card now):

```
bootc@ripley ~ $ gpg2 --edit-key 27703CF0
gpg (GnuPG) 2.0.20; Copyright (C) 2013 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Secret key is available.

pub  1024R/27703CF0  created: 2013-06-08  expires: never       usage: SC
                     trust: ultimate      validity: ultimate
sub  1024R/41320871  created: 2013-06-08  expires: 2014-06-08  usage: S
sub  1024R/B47AED2F  created: 2013-06-08  expires: 2014-06-08  usage: E
sub  1024R/4495E34E  created: 2013-06-08  expires: 2014-06-08  usage: A
[ultimate] (1). Chris Boot (TESTING ONLY) <bootc@bootc.net>
[ultimate] (2)  Chris Boot (STILL TESTING) <otheremail@bootc.net>

gpg> toggle
```

```
sec  1024R/27703CF0  created: 2013-06-08  expires: never
ssb  1024R/41320871  created: 2013-06-08  expires: never
ssb  1024R/B47AED2F  created: 2013-06-08  expires: never
ssb  1024R/4495E34E  created: 2013-06-08  expires: never
(1)  Chris Boot (TESTING ONLY) <bootc@bootc.net>
(2)  Chris Boot (STILL TESTING) <otheremail@bootc.net>

gpg> key 1

sec  1024R/27703CF0  created: 2013-06-08  expires: never
ssb* 1024R/41320871  created: 2013-06-08  expires: never
ssb  1024R/B47AED2F  created: 2013-06-08  expires: never
ssb  1024R/4495E34E  created: 2013-06-08  expires: never
(1)  Chris Boot (TESTING ONLY) <bootc@bootc.net>
(2)  Chris Boot (STILL TESTING) <otheremail@bootc.net>

gpg> keytocard
Signature key ....: [none]
Encryption key....: [none]
Authentication key: [none]

Please select where to store the key:
   (1) Signature key
   (3) Authentication key
Your selection? 1

You need a passphrase to unlock the secret key for
user: "Chris Boot (TESTING ONLY) <bootc@bootc.net>"
1024-bit RSA key, ID 41320871, created 2013-06-08

sec  1024R/27703CF0  created: 2013-06-08  expires: never
ssb* 1024R/41320871  created: 2013-06-08  expires: never
                     card-no: 0005 00001BD4
ssb  1024R/B47AED2F  created: 2013-06-08  expires: never
ssb  1024R/4495E34E  created: 2013-06-08  expires: never
(1)  Chris Boot (TESTING ONLY) <bootc@bootc.net>
(2)  Chris Boot (STILL TESTING) <otheremail@bootc.net>

gpg> key 1

sec  1024R/27703CF0  created: 2013-06-08  expires: never
ssb  1024R/41320871  created: 2013-06-08  expires: never
                     card-no: 0005 00001BD4
ssb  1024R/B47AED2F  created: 2013-06-08  expires: never
ssb  1024R/4495E34E  created: 2013-06-08  expires: never
(1)  Chris Boot (TESTING ONLY) <bootc@bootc.net>
(2)  Chris Boot (STILL TESTING) <otheremail@bootc.net>

gpg> key 2

sec  1024R/27703CF0  created: 2013-06-08  expires: never
ssb  1024R/41320871  created: 2013-06-08  expires: never
                     card-no: 0005 00001BD4
ssb* 1024R/B47AED2F  created: 2013-06-08  expires: never
ssb  1024R/4495E34E  created: 2013-06-08  expires: never
(1)  Chris Boot (TESTING ONLY) <bootc@bootc.net>
(2)  Chris Boot (STILL TESTING) <otheremail@bootc.net>
```

```
gpg> keytocard
Signature key ....: EC0B 5008 6494 A347 780D  E88C 2680 743C 4132 0871
Encryption key....: [none]
Authentication key: [none]

Please select where to store the key:
   (2) Encryption key
Your selection? 2

You need a passphrase to unlock the secret key for
user: "Chris Boot (TESTING ONLY) <bootc@bootc.net>"
1024-bit RSA key, ID B47AED2F, created 2013-06-08

sec  1024R/27703CF0  created: 2013-06-08  expires: never
ssb  1024R/41320871  created: 2013-06-08  expires: never
                     card-no: 0005 00001BD4
ssb* 1024R/B47AED2F  created: 2013-06-08  expires: never
                     card-no: 0005 00001BD4
ssb  1024R/4495E34E  created: 2013-06-08  expires: never
(1)  Chris Boot (TESTING ONLY) <bootc@bootc.net>
(2)  Chris Boot (STILL TESTING) <otheremail@bootc.net>

gpg> key 2

sec  1024R/27703CF0  created: 2013-06-08  expires: never
ssb  1024R/41320871  created: 2013-06-08  expires: never
                     card-no: 0005 00001BD4
ssb  1024R/B47AED2F  created: 2013-06-08  expires: never
                     card-no: 0005 00001BD4
ssb  1024R/4495E34E  created: 2013-06-08  expires: never
(1)  Chris Boot (TESTING ONLY) <bootc@bootc.net>
(2)  Chris Boot (STILL TESTING) <otheremail@bootc.net>

gpg> key 3

sec  1024R/27703CF0  created: 2013-06-08  expires: never
ssb  1024R/41320871  created: 2013-06-08  expires: never
                     card-no: 0005 00001BD4
ssb  1024R/B47AED2F  created: 2013-06-08  expires: never
                     card-no: 0005 00001BD4
ssb* 1024R/4495E34E  created: 2013-06-08  expires: never
(1)  Chris Boot (TESTING ONLY) <bootc@bootc.net>
(2)  Chris Boot (STILL TESTING) <otheremail@bootc.net>

gpg> keytocard
Signature key ....: EC0B 5008 6494 A347 780D  E88C 2680 743C 4132 0871
Encryption key....: CD2B 0337 67ED 8624 FEA8  D80F 0F8B 304C B47A ED2F
Authentication key: [none]

Please select where to store the key:
   (3) Authentication key
Your selection? 3

You need a passphrase to unlock the secret key for
user: "Chris Boot (TESTING ONLY) <bootc@bootc.net>"
1024-bit RSA key, ID 4495E34E, created 2013-06-08

sec  1024R/27703CF0  created: 2013-06-08  expires: never
```

```
ssb  1024R/41320871  created: 2013-06-08  expires: never
                     card-no: 0005 00001BD4
ssb  1024R/B47AED2F  created: 2013-06-08  expires: never
                     card-no: 0005 00001BD4
ssb* 1024R/4495E34E  created: 2013-06-08  expires: never
                     card-no: 0005 00001BD4
(1)  Chris Boot (TESTING ONLY) <bootc@bootc.net>
(2)  Chris Boot (STILL TESTING) <otheremail@bootc.net>

gpg> save
bootc@ripley ~ $
```

That's is folks, you now have one or two smart cards with your GnuPG keys on.

Creating a revocation certificate

It would be wise to create a revocation certificate for your new key. This is used should you lose access to the key completely so that you can revoke your key, e.g. tell the world not to use it any longer. You want to create this now, because if you lose your key you can no longer create a revocation certificate!

```
bootc@ripley ~ $ gpg2 --output /mnt/gpg-key-backup/revoke.asc --gen-revoke 27703CF0

sec  1024R/27703CF0 2013-06-08 Chris Boot (TESTING ONLY) <bootc@bootc.net>

Create a revocation certificate for this key? (y/N) y
Please select the reason for the revocation:
  0 = No reason specified
  1 = Key has been compromised
  2 = Key is superseded
  3 = Key is no longer used
  Q = Cancel
(Probably you want to select 1 here)
Your decision? 1
Enter an optional description; end it with an empty line:
> Generated at key creation time. Emergency use only.
> [ press enter ]
Reason for revocation: Key has been compromised
Generated at key creation time. Emergency use only.
Is this okay? (y/N) y

You need a passphrase to unlock the secret key for
user: "Chris Boot (TESTING ONLY) <bootc@bootc.net>"
1024-bit RSA key, ID 27703CF0, created 2013-06-08

ASCII armored output forced.
Revocation certificate created.

Please move it to a medium which you can hide away; if Mallory gets
access to this certificate he can use it to make your key unusable.
It is smart to print this certificate and store it away, just in case
your media become unreadable.  But have some caution:  The print system of
your machine might store the data and make it available to others!
```

You need to make sure you can never lose access to the revocation certificate, after all you'll need it in an emergency, but also make sure it cannot fall into the wrong hands. With this certificate, anyone can revoke your key.

Exporting the public key

Export the public portion of the key to a file now. We'll use this in our daily GnuPG keyring to test with, and upload it to the key servers later once we're happy everything is OK. This part of the key is public, and you can and should spread it far and wide.

```
bootc@ripley ~ $ gpg2 --armor --export 27703CF0 > 27703CF0.asc
bootc@ripley ~ $
```

Tidying up, securing the backup

Let's tidy up and lock up our backup volume now:

```
bootc@ripley ~ $ rm -rf /mnt/gpg-key-backup/gnupghome-temp/
bootc@ripley ~ $ sudo -s
[sudo] password for bootc:
ripley bootc # umount /mnt/gpg-key-backup
ripley bootc # cryptsetup luksClose gpg-key-backup
ripley bootc # exit
bootc@ripley ~ $ unset GNUPGHOME
bootc@ripley ~ $
```

Testing

Now, we'll import the public key into our normal `~/.gnupg` directory, and set it up for use:

```
bootc@ripley ~ $ gpg2 --import 27703CF0.asc
gpg: keyring `/home/bootc/temp-gpg/secring.gpg' created
gpg: keyring `/home/bootc/temp-gpg/pubring.gpg' created
gpg: /home/bootc/temp-gpg/trustdb.gpg: trustdb created
gpg: key 27703CF0: public key "Chris Boot (TESTING ONLY) <bootc@bootc.net>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
bootc@ripley ~ $ gpg2 --card-status
Application ID ...: D2760001240102000005000001BD40000
Version ..........: 2.0
Manufacturer .....: ZeitControl
Serial number ....: 00001BD4
Name of cardholder: [not set]
Language prefs ...: de
Sex ..............: unspecified
URL of public key : [not set]
Login data .......: [not set]
Signature PIN ....: forced
Key attributes ...: 1024R 1024R 1024R
Max. PIN lengths .: 32 32 32
PIN retry counter : 3 0 3
Signature counter : 0
Signature key ....: EC0B 5008 6494 A347 780D  E88C 2680 743C 4132 0871
      created ....: 2013-06-08 18:22:52
Encryption key....: CD2B 0337 67ED 8624 FEA8  D80F 0F8B 304C B47A ED2F
      created ....: 2013-06-08 18:38:04
Authentication key: 7429 DE50 66FA B50B 7833  F173 9C96 F566 4495 E34E
      created ....: 2013-06-08 18:38:25
General key info..: pub  1024R/41320871 2013-06-08 Chris Boot (TESTING ONLY)
↪<bootc@bootc.net>
```

```
sec#  1024R/27703CF0  created: 2013-06-08  expires: never
ssb>  1024R/41320871  created: 2013-06-08  expires: 2014-06-08
                      card-no: 0005 00001BD4
ssb>  1024R/B47AED2F  created: 2013-06-08  expires: 2014-06-08
                      card-no: 0005 00001BD4
ssb>  1024R/4495E34E  created: 2013-06-08  expires: 2014-06-08
                      card-no: 0005 00001BD4
bootc@ripley ~ $
```

Notice when we ran card-status this time, it printed out the key info for the sub-keys that are on the card. The `#` after the `sec` means there is no private key available for our primary key, this is good. The `>` after each `ssb` and the `card-no` listed means those private keys are available on the smart card, this is also good.

Now we need to mark our key as trusted. Just because we have the secret keys doesn't mean we should trust the key automatically, so we have to explicitly tell GnuPG that we trust this key, because it's our own:

```
bootc@ripley ~ $ gpg2 --edit-key 27703CF0
gpg (GnuPG) 2.0.20; Copyright (C) 2013 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Secret key is available.

pub  1024R/27703CF0  created: 2013-06-08  expires: never       usage: SC
                     trust: unknown       validity: unknown
sub  1024R/41320871  created: 2013-06-08  expires: 2014-06-08  usage: S
sub  1024R/B47AED2F  created: 2013-06-08  expires: 2014-06-08  usage: E
sub  1024R/4495E34E  created: 2013-06-08  expires: 2014-06-08  usage: A
[ unknown] (1). Chris Boot (TESTING ONLY) <bootc@bootc.net>
[ unknown] (2)  Chris Boot (STILL TESTING) <otheremail@bootc.net>

gpg> trust
pub  1024R/27703CF0  created: 2013-06-08  expires: never       usage: SC
                     trust: unknown       validity: unknown
sub  1024R/41320871  created: 2013-06-08  expires: 2014-06-08  usage: S
sub  1024R/B47AED2F  created: 2013-06-08  expires: 2014-06-08  usage: E
sub  1024R/4495E34E  created: 2013-06-08  expires: 2014-06-08  usage: A
[ unknown] (1). Chris Boot (TESTING ONLY) <bootc@bootc.net>
[ unknown] (2)  Chris Boot (STILL TESTING) <otheremail@bootc.net>

Please decide how far you trust this user to correctly verify other users' keys
(by looking at passports, checking fingerprints from different sources, etc.)

  1 = I don't know or won't say
  2 = I do NOT trust
  3 = I trust marginally
  4 = I trust fully
  5 = I trust ultimately
  m = back to the main menu

Your decision? 5
Do you really want to set this key to ultimate trust? (y/N) y

pub  1024R/27703CF0  created: 2013-06-08  expires: never       usage: SC
                     trust: ultimate      validity: unknown
sub  1024R/41320871  created: 2013-06-08  expires: 2014-06-08  usage: S
sub  1024R/B47AED2F  created: 2013-06-08  expires: 2014-06-08  usage: E
sub  1024R/4495E34E  created: 2013-06-08  expires: 2014-06-08  usage: A
```

```
[ unknown] (1). Chris Boot (TESTING ONLY) <bootc@bootc.net>
[ unknown] (2)  Chris Boot (STILL TESTING) <otheremail@bootc.net>
Please note that the shown key validity is not necessarily correct
unless you restart the program.

gpg> save
Key not changed so no update needed.
```

Now remove the card, to prove we can't do sensitive operations without the card. We'll perform some tests:

```
bootc@ripley ~ $ echo "this is signed" > signed.txt
bootc@ripley ~ $ echo "this is encrypted" > encrypt.txt
bootc@ripley ~ $ gpg2 -r bootc@bootc.net -ae encrypt.txt
bootc@ripley ~ $ gpg2 -s signed.txt
gpg: selecting openpgp failed: Card not present
gpg: signing failed: Operation cancelled
gpg: signing failed: Operation cancelled
bootc@ripley ~ $ gpg2 -d encrypt.txt.asc
gpg: selecting openpgp failed: Card not present
gpg: encrypted with 1024-bit RSA key, ID B47AED2F, created 2013-06-08
      "Chris Boot (TESTING ONLY) <bootc@bootc.net>"
gpg: public key decryption failed: Operation cancelled
gpg: decryption failed: No secret key
bootc@ripley ~ $
```

Note that you can encrypt data with only access to the public keys (but you can't decrypt it), and you cannot sign either. You would be able to verify signatures, however. Now let's insert the card and try the decryption and signing again:

```
bootc@ripley ~ $ gpg2 -d encrypt.txt.asc
gpg: encrypted with 1024-bit RSA key, ID B47AED2F, created 2013-06-08
      "Chris Boot (TESTING ONLY) <bootc@bootc.net>"
this is encrypted
bootc@ripley ~ $ gpg2 -s signed.txt
bootc@ripley ~ $ gpg2 --verify signed.txt.gpg
gpg: Signature made Sat 08 Jun 2013 20:58:06 BST using RSA key ID 41320871
gpg: Good signature from "Chris Boot (TESTING ONLY) <bootc@bootc.net>"
gpg:                 aka "Chris Boot (STILL TESTING) <otheremail@bootc.net>"
bootc@ripley ~ $
```

That's it!

CHAPTER 2

Indices and tables

- genindex
- search