An Advanced Introduction to GnuPG

Neal H. Walfield

August 15, 2017

Contents

Part I

Main Matter

Chapter 1

Key Creation

Today, creating an OpenPGP key could hardly be easier or less error prone. It's as simple as thinking of a password and using `gpg`'s `--quick-gen-key` command:

```
$ gpg --quick-gen-key 'Juliet Capulet <juliet@gnupg.net>'
About to create a key for:
    "Juliet Capulet <juliet@gnupg.net>"

Continue? (Y/n) y
...
gpg: revocation certificate stored as
 '/home/jc/.gnupg/openpgp-revocs.d/98DB84C56F56DB5CF4733CCDEACAE136B8AF8CC2.re
public and secret key created and signed.

pub   rsa2048 2017-08-11 [SC] [expires: 2019-08-11]
      98DB84C56F56DB5CF4733CCDEACAE136B8AF8CC2
uid                      Juliet Capulet <juliet@gnupg.net>
sub   rsa2048 2017-08-11 [E]
```

(The above confirmation prompt can be suppressed by including the `--batch` option, which, as its name suggests, is designed for batch operations. For batch operations, the raw output of `gpg` shouldn't be parsed, but, instead, `--status-fd` should be used to get a stable interface. Also, in this case, `--pinentry-mode loopback --passphrase-fd X` can be used to supply a password.)

If you have multiple email addresses, then it is useful to also add them to your key if they are in the same trust domain. (For instance, work and private email should often be kept separate.) This can be done just as painlessly using the `--quick-add-uid` option:

```
$ gpg --quick-add-uid 98DB84C56F56DB5CF4733CCDEACAE136B8AF8CC2 \
> 'Juliet Capulet <juliet@riseup.net>'
```

For most users, the only important thing left to do is to backup the revocation certificate (this is explained in the next section), and publish the key, so that others can find it:

```
$ gpg --send-key 98DB84C56F56DB5CF4733CCDEACAE136B8AF8CC2
gpg: sending key EACAE136B8AF8CC2 to hkps://hkps.pool.sks-keyservers.ne
```

For users with stronger security requirements—those users who are not just worried about protecting their privacy—we recommend that they use a security token instead of an online key so that if the device is compromised, an attacker cannot get access to the secret key material. This is actually very easy: any program can normally access any file. Thus, using an online key, it is necessary to trust *all* programs that run on your system. Although setting up a security token—the focus of Section 1.3—is more work, the day-to-day use of a security token is no more complicated than an online key.

If you are replacing an existing key, then it is strongly recommended that you have the old key sign the new one, and the new key sign the old one so that there is strong, machine readable evidence that the two keys are controlled by the same party. This information is used by `gpg` in the TOFU trust model, for instance, to avoid spurious conflicts.

1.1 Keys Aren't Forever, Revocation Certificates Are

There are several reasons for why you might want to create a new key.

The most security relevant reason is that your current key could be compromised. This is the case if, for example, your device was infected with malware, or it was lost or stolen. In such cases, it is essential to immediately inform your communication partners that any messages encrypted to your key—not only new messages, but also messages encrypted prior to

the compromise—could be read by a third party, and that signatures made by your key should no longer be trusted.

A less acute, but still important security relevant reason for creating a new key is that your old key no longer satisfies your security requirements. This could either be because your security requirements have changed, or the key has become weaker due to advances in cryptanalysis (the science of breaking cryptographic systems). An example of the latter is that 1024-bit RSA keys are no longer considered sufficiently strong to stop nation-state attackers. In reaction to this development, organizations like Debian now require members who have such keys to generate new ones. Although the immediate security implications are not as grave as above (unless you really think you are being targeted by a nation-state adversary), communication partners still need to be told to start using the new key.

A practical reason for creating a new key is that your old one has become inaccessible. This can happen if you forget your password, or you forget to migrate your key when reinstalling your system. Novice users are often beset by these problems. This usually leads to mails that can't be decrypted, because someone used the old key, which still appears to be valid. This not only inconveniences the recipient, because she can't decrypt the email, but also the sender, because the recipient will have to ask him to resend the email using her new key. This mistake appears to the users as a usability problem. But, it is worse; it is a security problem: users learn that encrypting email is fragile. And, taking this lesson to heart, they will reserve encryption for messages that "really" need protection. But, when the time comes, these users will be out of practice, which increases the chance that they mistakenly leave some important data unencrypted.

There is only one way to deal with these issues: the user needs to somehow inform her communication partners that her old key is no longer valid, and that a new one should be used instead. This can be done by talking to each person. But, this is inconvenient for everyone involved. Instead, it is easier, and less error prone to simply codify this into the system, which is what a revocation certificate does: it states that a particular key is no longer valid. And, a user's communication partners don't normally have to take any special actions: once uploaded to a key server, it will automatically be respected the next time the software refreshes the key.

A revocation certificate is only considered valid if it includes a self-signature. This presents a problem for users who lost access to their key: they can't create a revocation certificate! To mitigate this prob-

lem, version 2.1 of GnuPG automatically creates a revocation certificate when creating a new key. (The revocation certificate is stored in the `$GNUPGHOME/openpgp-revocs.d` directory.)

Because `gpg` doesn't know how the certificate will be used, it creates a generic revocation certificate. In some cases, it is useful to provide more details. But, in practice, this is rarely necessary: most users won't see the reason, and GnuPG treats the different reasons in the same way. Other OpenPGP implementations appear to do the same thing.

Automatically creating the revocation certificate when the key is created ensures that users who forget their password can still revoke their key. But, this doesn't help users who accidentally delete their key, e.g., by reinstalling their system, or forgetting to migrate it to a new computer. To protect against this mistake, it is essential that the revocation certificate be backed up on a separate system.

1.1.1 Backing Up a Revocation Certificate

To help ensure that the revocation certificate remains accessible when it is finally needed, it should be stored in multiple places, but not someplace that is easily accessible to an adversary. This is hard to do automatically.

One easy way to make a backup is to store the revocation certificate on the user's mail server. Of course, anyone with access to the mail account, such as the mail provider, could publish it. But, this is unlikely to happen in practice, and the consequences are more an inconvenience than a security issue: the user has to create a new key, and should tell her contacts what happened; the attacker is not able to forge signatures, or decrypt any messages.

Another way to make a backup is to display the revocation certificate as a QR code, and have the user photograph it. On Debian, this can be done using `qrencode`:

```
$ apt-get install qrencode
$ qrencode -o $FINGERPRINT.png < $FINGERPRINT.rev
$ eog $FINGERPRINT.png
```

Even if the user doesn't have a QR code scanner installed on her camera, it is possible to decode it later. This can be done, for instance, using ZBar, which is available on Debian as part of the `zbar-tools` package. Because the user probably won't know what the QR code is for after a few weeks,

it is essential to add text next to the QR code to explain that the QR code contains a revocation certificate. The main security problem here is that many phones automatically backup data to the cloud, and, as above, the provider needs to be trusted to not publish it.

It is also reasonable to print the revocation certificate. Paper, for instance, has much better archival properties than many digital storage mediums, such as CD-ROMs. This can either be in text form (but this form is a pain to reenter) or as a QR code. Unfortunately, printers are no longer as secure as they once were: to simplify printing, some local printers are accessed via the cloud! But, even if this is not the case, in the very least, most are connected to the internet, and don't receive software updates. But, as before, the damage that an attacker who has a revocation certificate can cause is limited.

1.1.2 Publishing a Revocation Certificate

A key isn't really revoked until all communication partners have a copy of the revocation certificate. The easiest way to accomplish this is to import the revocation certificate locally, and then publish it on the public key servers.

```
$ gpg --import 98DB84C56F56DB5CF4733CCDEACAE136B8AF8CC2.rev
gpg: key EACAE136B8AF8CC2: "Juliet Capulet <juliet@riseup.net>" revocation cer
gpg: Total number processed: 1
gpg:    new key revocations: 1
$ gpg --send-key 98DB84C56F56DB5CF4733CCDEACAE136B8AF8CC2
gpg: sending key EACAE136B8AF8CC2 to hkps://hkps.pool.sks-keyservers.net
```

When using the revocation certificate that `gpg` automatically generated at key creation time, one more step is required: to prevent the user from accidentally importing the revocation certificate, GnuPG requires that the user first edit the file to remove a comment character. The contents of the file explain this, but importing the file does not provide a helpful error message (`gpg` just indicates that the file contains "no valid OpenPGP data"), and most users won't know to look at the files, which explains what to do:

```
...
To avoid an accidental use of this file, a colon has been inserted
before the 5 dashes below.  Remove this colon with a text editor
```

```
before importing and publishing this revocation certificate.

:-----BEGIN PGP PUBLIC KEY BLOCK-----
Comment: This is a revocation certificate

iQE2BCABCAAgFiEEmNuExW9W21z0czzN6srhNrivjMIFAlmNo+4CHQAACgkQ6srh
...
```

Assuming the user's communication partners have configured their software to automatically refresh keys, this should be enough. But it is nevertheless recommended that the updated key be sent to frequent communication partners to ensure a timely notification. This is best done by attaching a minimal key, which can be generated as follows:

```
$ gpg -a --export --export-options export-minimal \
> 98DB84C56F56DB5CF4733CCDEACAE136B8AF8CC2 > juliet-key.gpg
```

1.1.3 Recruiting Your Friends

Another way to deal with the revocation problem is to make a trusted third party a so-called *designated revoker*. This can be done by using the `addrevoker` subcommand in the `--edit-key` interface. For instance, Romeo could designate Juliet as a revoker for his key:

```
$ gpg --edit-key romeo@gnupg.net
Secret key is available.

sec  rsa2048/B003B1463C7B41BE
     created: 2017-08-11  expires: 2019-08-11  usage: SC
     trust: ultimate      validity: ultimate
ssb  rsa2048/50A1A6C84DBDEA6F
     created: 2017-08-11  expires: never       usage: E
[ultimate] (1). Romeo Montague <romeo@gnupg.net>

gpg> addrevoker

Enter the user ID of the designated revoker: 98DB84C56F56DB5CF4733CCDE
pub  rsa2048/EACAE136B8AF8CC2 2017-08-11 Juliet Capulet <juliet@riseup
 Primary key fingerprint: 98DB 84C5 6F56 DB5C F473  3CCD EACA E136 B8A
```

```
WARNING: appointing a key as a designated revoker cannot be undone!

Are you sure you want to appoint this key as a designated revoker? (y/N) y

This key may be revoked by RSA key EACAE136B8AF8CC2 Juliet Capulet <juliet@ris
sec  rsa2048/B003B1463C7B41BE
     created: 2017-08-11  expires: 2019-08-11  usage: SC
     trust: full          validity: full
ssb  rsa2048/50A1A6C84DBDEA6F
     created: 2017-08-11  expires: never       usage: E
[  full  ] (1). Romeo Montague <romeo@gnupg.net>

gpg> quit
Save changes? (y/N) y
```

Now, Juliet can generate a revocation certificate for Romeo's key using `gpg` 's `--generate-designated-revocation` command. The resulting revocation certificate is just like a normal revocation certificate. So, as above, Juliet would import this certificate, and then publish Romeo's key to cause it to be revoked.

1.2 Tweaking, Twiddling, and Frobbing

Prior to GnuPG 2.1, the `--quick-gen-key` command did not exist. To generate a key, users instead had to use the `--gen-key` command. The `--gen-key` command is like `--quick-gen-key`, but prompts the user to set a number of parameters. (This command, as well as its even more flexible variant, `--full-key-gen`, are still available.) But, in practice, few users need to use anything but the defaults. In fact, the defaults are even reasonable for almost all experts. Unfortunately, because the `--gen-key` command asks for user input, users appear to assume that the defaults need to be tweaked. And, this idea that the defaults are not sufficient is further reinforced by most GnuPG guides that suggest "better," more "secure" settings. In almost all cases, these guides only confuse, overwhelm, and scare users.

Now, it is almost certainly true that a larger key is harder to brute force than a smaller key. That is, a larger key increases the strength of the cryp-

tography. But, larger key sizes have a cost. In particular, verifying a signature generated by a 4096-bit RSA key doesn't take twice as long as a 2048-bit RSA key, but orders of magnitude longer. So, the question is: does the stronger cryptography actually increase the system's security? Bruce Schneier argues that the Snowden leaks provide strong evidence that the NSA has not broken strong cryptography: when the NSA wants to access someone's data, they compromise the infrastructure and the endpoints, which is more costly, and more likely to be noticed [1]. Assuming Schneier is correct, since the strongest potential adversary in the world isn't breaking strong cryptography, further increasing the strength of the cryptography will *not* increase the system's security. Instead, to increase the system's security, it is better to protect the endpoint, and improve the user's operational security.

There are several things that can be done to better protect an endpoint. These include encrypting the storage device, using a good password for logging in, enabling a screen locker, promptly installing system updates, and avoiding malware. The next step is to better protect the cryptographic keys. This can be done by using a security token, which is a small piece of hardware that stores the cryptographic keys, and performs the basic cryptographic operations. In this way, if the end point is compromised, the keys are still safe. Although any data that was decrypted while the adversary controlled the computer could have been recovered.

For those few cases where it is necessary to override some defaults, it is still possible to use the `--key-gen` command or `--full-key-gen` command. And, it is also possible to modify a key using `gpg` 's `--edit-key` interface.

`gpg` also provides an interface for batch operations. See the "Unattended key generation" chapter of the GnuPG manual for details.

1.3 Security Tokens

A security token is a small piece of hardware that is typically connected to a computer via USB or NFC. The security token holds the keys and performs the primitive crypto operations, such as, decrypting a session key, or generating a signature. In this way, the secret key material never needs to be on the internet-connected device.

Not having the secret key material on the main device has a number

of advantages: if an adversary wants to decrypt or sign messages, it is not sufficient to compromise the endpoint and exfiltrate the keys; the attacker needs to maintain a presence on the device, and wait for the user to insert the security token to decrypt or sign a message. For smartcard readers with an integrated PIN pad, the attacker also has to convince the user to enter the security token's PIN. This is a great defense, because although a user might enter the PIN a few times, most people will quickly become suspicious of many spurious prompts. Unfortunately, readers with an integrated PIN pad are bulky, which makes them inconvenient for mobile users. But, even without a PIN pad, security tokens significantly increase security. Of course, a security token can't prevent an attacker who has control over the endpoint from seeing any messages that are decrypted on the device.

A security token also has the advantage that if the device is compromised, it is not necessary to completely revoke the OpenPGP key. At most the signing subkey needs to be rotated (see Section 1.5), if the attacker might have signed a message. This is a great advantage, because the user's fingerprint stays the same, and certifications remain valid. Normally, creating a new OpenPGP key means that the user not only has to inform all of her contacts that she has a new key, and print new business cards, but she also has to recertify all of the keys that she signed, and convince people who signed her old key to sign her new one.

Given their security properties, and the relative ease of use once they are setup, we strongly recommend that anyone who relies on GnuPG to protect their security use a security token. Using an online key is reasonable for people who use GnuPG to protect their privacy or protect themselves from phishing excursions, or who want to hinder mass surveillance.

1.3.1 Hardware

There are a variety of security tokens that support OpenPGP. These have different advantages and disadvantages in terms of the degree to which they respect the user's freedom, their security properties, their feature sets, and their commercial availability. Below, we introduce a few that are known to work well with GnuPG.

OpenPGP Smartcard

The OpenPGP smartcard has been around since 2003. Although the software is proprietary, the specification is freely available and usable without constraints [2], and it has become the de facto interface for interacting with OpenPGP security tokens.

The main distributor is Kernel Concepts. They sell them to end users in their online shop, and they ship worldwide (`https://www.floss-shop.de/en`). These smartcards are relatively inexpensive. At the time of this writing, the FLOSS Shop sells them for 16.40 euros. But, because most systems don't include a smartcard reader, this hardware also needs to be purchased. Depending on the required features, in particular, whether an external PIN entry pad is desired, a smartcard reader currently costs between 20 euros and 50 euros.

Gnuk

The Gnuk security token (`https://www.fsij.org/category/gnuk.html`) was created by NIIBE Yutaka in 2012. His goal is to create a security token that not only uses free software, but whose schematics are freely available, and whose parts can be sourced by anyone. These constraints mean that the Gnuk does not use specialized security hardware that has been strengthened to prevent physical key extraction attacks, because this hardware's documentation is typically only available by signing an NDA, and can not be easily bought by individuals. But, although commodity hardware can't protect the user from key extraction attacks half as well as specially hardened hardware, it is much more difficult for an attacker to introduce a backdoor on all versions of the product. For instance, the MCU that Gnuk uses is produced by many manufacturers.

Using an alternate official firmware, the Gnuk can also function as a true random number generator (TRNG).

Nitrokey

The Nitrokey (`https://www.nitrokey.com/`) is based on the Gnuk, but is more feature rich. For instance, it supports OTP and U2F. And, some versions have on-board storage. The code is open source, but the schematics are not freely available.

YubiKey

YubiKey (`https://yubikey.com`) is a popular security token that has been adopted by some large organizations including Google and GitHub, and is sold by many major retailers. Like the Nitrokey, YubiKeys support several different security systems. But, not all YubiKeys support the OpenPGP card interface: support for this functionality is in the current product line is limited to the YubiKey NEO, and the YubiKey 4. The YubiKey NEO also supports NFC. This wireless interface makes it easy to use the YubiKey with a mobile phone. And, OpenKeychain, which is an OpenPGP implementation that integrates with the K-9 mailer on Android, supports it.

Like the OpenPGP smartcard, the YubiKey's firmware is proprietary. YubiKey used to release their OpenPGP applet as open source, but decided that due to their focus on security, and the inability to reflash the devices, making the source code available provides "little practical value" [3].

1.3.2 Creating a Key

Most security tokens include functionality to create an OpenPGP key. But, using this functionality is dangerous. First, given the limited hardware, and the typically proprietary software, the quality of the entropy is questionable. Unfortunately, the importance of high quality entropy for the security of the system can not be understated. For instance, the NSA is known to have backdoored the Dual_EC_DRBG pseudorandom number generator, a standard adopted by NIST, to allow them to recover encryption keys, and paid companies like RSA to make it the default in their products [4]. Second, security tokens do not provide an option to export keys. This limitation is highly desirable to prevent an attacker from recovering keys if the device is stolen or lost. But, it also means that it is not possible to backup the keys. Since OpenPGP is used to protect long-lived data, this is a serious limitation.

The alternative to creating an OpenPGP key on the security token is to create it on a computer, and then copy it to the security token. GnuPG supports this out of the box. But, with some distributions, such as Debian, GnuPG's smartcard support is packaged separately and not installed by default. On Debian and Ubuntu, GnuPG's smartcard support is included in the `scdaemon` package. Depending on the security token, it may also be

necessary to install `pcscd`, which includes additional drivers.

Using An Offline Computer

If you are going to take the trouble to use a smartcard to separate your secret keys and your main system, then you can't use your main system to manage your keys.

There are two ways around this: you can either use a dedicated offline computer, or a live CD. The former is more secure, particularly, if you are willing to completely cut off its network access (i.e., air gap it by removing all network cards, bluetooth module, etc.). Given that used IBM Thinkpads are readily available for under 50 euros, this is the preferred solution. That said, for the majority of people who have modest security requirements, managing keys from a live CD that is booted from their main computer is reasonable.

Unfortunately, most live CDs are not appropriate for working with offline keys. Tails, however, is a GNU/Linux distribution that takes the necessary precautions. First, Tails starts as few services as possible to reduce the attack surface, and, with one clearly marked exception, it doesn't allow outbound network access except over Tor. And, when Tails shuts down, it wipes the system's memory. This is essential to make sure the keys are not accidentally exposed to an attacker. This protection is not just necessary to prevent cold boot attacks [5], which only those who require the highest levels of security have to worry about, but also to prevent the system after it restarts from accessing, and perhaps accidentally leaking the keys in uninitialized memory.

Whether you use a dedicated computer or reboot your normal computer into Tails, you need:

- A security token,
- The Tails distribution,
- A bootable storage device for Tails (at least 2 GB large),
- A storage device for your secret key material, and
- A storage device for exchanging files with your main system.

It doesn't matter whether the storage devices are USB keys, SD cards, hard drives, or something else. The important bit is that your computer can

boot from the one for Tails, and the security token, and the storage devices can all be attached to the computer at the same time.

1.3.3 Tails

Tails is available from `https://tails.boum.org/`. The key used to sign the distribution is:

```
A490 D0F4 D311 A415 3E2B  B7CA DBB8 02B2 58AC D84F
```

First, download the ISO image, and the corresponding signature file. The latest version is linked to from `https://tails.boum.org/install/download/openpgp/`, and the files are named like `tails-amd64-VERSION.iso` and `tails-amd64-VERSION.iso.sig`.

Next, verify the signature. To do this, you need to first fetch the aforementioned key:

```
$ gpg --recv-key A490D0F4D311A4153E2BB7CADBB802B258ACD84F
gpg: requesting key 58ACD84F from hkp server keys.gnupg.net
gpg: key 58ACD84F: public key "Tails developers (offline long-term identity ke
gpg: no ultimately trusted keys found
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
$ gpg --verify tails-amd64-VERSION.iso.sig tails-amd64-VERSION.iso
gpg: Signature made Wed 09 Aug 2017 02:06:36 AM CEST
gpg:                using RSA key 79192EE220449071F589AC00AF292B44A0EDAA41
                                  ----------------------------------------
gpg: Good signature from "Tails developers (offline long-term identity key) <t
gpg:                 aka "Tails developers <tails@boum.org>" [undefined]
```

The `--verify` command shows us that the signature is "good" in the sense that the signature was really over the ISO image. But, it also shows us that the signature was generated by the key `79192EE220449071F589AC00AF292B44A0EDAA41`, not by the key `A490D0F4D311A4153E2BB7CADBB802B258ACD84F`! A close examination reveals that this is because the Tails developers used a signing subkey to make the signature. We can see this by using the `--list-keys` command to show more details about the key used to create the signature:

```
$ gpg -k 79192EE220449071F589AC00AF292B44A0EDAA41
pub   rsa4096/0xDBB802B258ACD84F 2015-01-18 [C] [expires: 2018-01-11]
      Key fingerprint = A490 D0F4 D311 A415 3E2B  B7CA DBB8 02B2 58AC D
                        ----------------------------------------
uid                  [  undef ] Tails developers (offline long-term ic
uid                  [  undef ] Tails developers <tails@boum.org>
sub   rsa4096/0x98FEC6BC752A3DB6 2015-01-18 [S] [expires: 2018-01-11]
sub   rsa4096/0x3C83DCB52F699C56 2015-01-18 [S] [expires: 2018-01-11]
sub   rsa4096/0xAF292B44A0EDAA41 2016-08-30 [S] [expires: 2018-01-11]
```

Note: when you try this, you might see a different signing key. This is okay. What is important is that the main key is correct in the sense that the fingerprint matches: the user ID is not enough to prove the download's authenticity; creating a key with an arbitrary user ID, and uploading it to a key server is easy. By rotating keys, the Tails developers can reduce the amount of time that an undetected compromise of the signing key is useful to an attacker.

If `gpg --verify` indicates that the signature is bad, or `gpg -k` indicates that the signing key is not associated with `A490D0F4D311A4153E2BB7CADBB802B258AC` then there is a problem with the download. You should first try again from a different network. If the problem persists, seek help from an expert. Since Tails is used by people who are trying to protect sensitive information, there are bound to be copies that have been modified to include malware; **do not use the ISO image if you (or someone you trust) can't verify it**.

Assuming the data is okay, it is now possible to copy the Tails ISO to a USB key. This can be done using `dd`:

```
$ sudo dd if=tails-amd64-VERSION.iso of=/dev/sdX bs=4096
```

(`sdX` should be replaced by the name of the actual storage device.)

Then, boot into Tails.

Before logging it, you can set the `root` password so that you can, for instance, install additional (optional) software. You can do this at the Tails login screen by going to *Additional Settings » Administration Password*. Note: if you are worried about a targeted attack, you should not connect the computer to the network, and this step can be skipped.

The Tails website has alternate installation and verification instructions. It is reasonable to follow them. Particularly, if you don't have a Unix-

like system with `gpg` already installed, which these instructions take for granted.

1.3.4 Initializing the Security Token

Once Tails has started, the first thing to do is to make sure that it recognizes the security token. Insert the security token, and then issue `gpg` 's `--card-status` command. The output should be similar to the following:

```
$ gpg --card-status
Reader ...........: 04E6:E003:21251019201732:0
Application ID ...: D2760001240102010005000002D9D0000
Version ..........: 2.1
Manufacturer .....: ZeitControl
Serial number ....: 00002D9D
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 ...: rsa2048 rsa2048 rsa2048
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]
```

Looking at the `Manufacturer` field, we see that the security token is an OpenPGP smartcard. Specifically, we know from the `Version` field that it implements version 2.1 of the OpenPGP card specification. Like all OpenPGP smartcards, it supports three keys: a signing key, an encryption key, and an authentication key. The `Signature key`, `Encryption key` and `Authentication key` fields tell us that no keys are currently stored on the card. The `Key attributes` field indicates that this particular

OpenPGP card supports up to 2048-bit RSA keys. Newer versions of the card support 4096-bit RSA keys.

If the card has already been used, then it first needs to be reset. This can be done using `--card-edit`'s `factory-reset` subcommand. If the factory reset command indicates that the card is not supported, as below, you may need to consult your security token's documentation:

```
$ gpg --card-edit
...
gpg/card> admin
Admin commands are allowed

gpg/card> factory-reset
gpg: OpenPGP card no. D2760001240102000FFFE50FF6C060000 detected
gpg: This command is not supported by this card
```

But, the following commands usually work:

```
$ gpg-connect-agent
/hex
scd apdu 00 20 00 81 08 40 40 40 40 40 40 40 40
scd apdu 00 20 00 81 08 40 40 40 40 40 40 40 40
scd apdu 00 20 00 81 08 40 40 40 40 40 40 40 40
scd apdu 00 20 00 81 08 40 40 40 40 40 40 40 40
scd apdu 00 20 00 83 08 40 40 40 40 40 40 40 40
scd apdu 00 20 00 83 08 40 40 40 40 40 40 40 40
scd apdu 00 20 00 83 08 40 40 40 40 40 40 40 40
scd apdu 00 20 00 83 08 40 40 40 40 40 40 40 40
scd apdu 00 e6 00 00
scd apdu 00 44 00 00
```

The first four lines after the `/hex` directive enter a bad user PIN. The next four enter a bad admin PIN. Then, the last two lines terminate, and reactivate the card, respectively. After removing the security token and reconnecting it, it should be reset. You can verify this by running `gpg --card-edit`, checking that the card contains no keys, and then using the `verify` subcommand to make sure the user PIN has been reset to the default (normally `123456`).

Next, we need to change the default PINs. The OpenPGP card has two PINs: a user PIN and an admin PIN. The user PIN is used on a day-to-day basis to authorize decryption and signing; the admin PIN allows adding new keys, among other things. The admin PIN should not be used on the main computer, and it should be different from the user PIN. The default PINs are usually `123456` and `12345678`, respectively.

To change the PINs, we need to first enable admin mode. Then, we can use the `passwd` command to change each PIN. GnuPG will prompt you to enter the old PIN and the new PIN. If you are using a PIN pad, this can be confusing: you need to enter the new PIN twice.

```
$ gpg --card-edit
...
gpg/card> admin
Admin commands are allowed

gpg/card> passwd
gpg: OpenPGP card no. D2760001240102010005000002D9D0000 detected

1 - change PIN
2 - unblock PIN
3 - change Admin PIN
4 - set the Reset Code
Q - quit

Your selection? 1
PIN changed.

1 - change PIN
2 - unblock PIN
3 - change Admin PIN
4 - set the Reset Code
Q - quit

Your selection? 3
PIN changed.

1 - change PIN
```

```
2 - unblock PIN
3 - change Admin PIN
4 - set the Reset Code
Q - quit

Your selection? q
```

You'll almost certainly want to write the admin PIN down someplace. Given how often it is normally used, most people forget it. A good place to hide it is at a physically different location from where you hide the USB key with your secret keys on it.

1.3.5 Formatting the Removable Storage Devices

As mentioned above, we need to format two storage devices: one for the secret keys, and one for the public keys. Assuming the partition holding the private key material is encrypted with a strong passphrase, it is reasonable to just have two partitions on a single memory card. But, if possible, it is better to never insert the USB key with the secret key material into an insecure computer.

The Encrypted File System

When we create the OpenPGP key, we need to store it somewhere. Although it is possible to install Tails and create a persistent volume, it is better to use a separate storage device. This way, upgrading your Tails "installation" is trivial: you just need to download a new live CD, and copy it to your dedicated Tails device; there is no data to migrate.

The amount of storage space for secret keys material doesn't need to be large: 10 megabytes is more than sufficient. But, it should be encrypted. The easiest way to create an encrypted file system in Tails is to use GNOME Disks (*Applications » Utilities » Disks*).

In GNOME Disks, select the USB key, create a new volume, set the label to "secret-keys", and format it as a "LUKS + Ext4" file system. See Figure 1.1. GNOME Disks does not automatically mount the new file system. But, you can do this by selecting the partition, and clicking on the "play" button. The file system will be mounted under `/media/amnesia/secret-keys`.

You'll probably also want to write this passphrase on the piece of paper with your admin PIN.

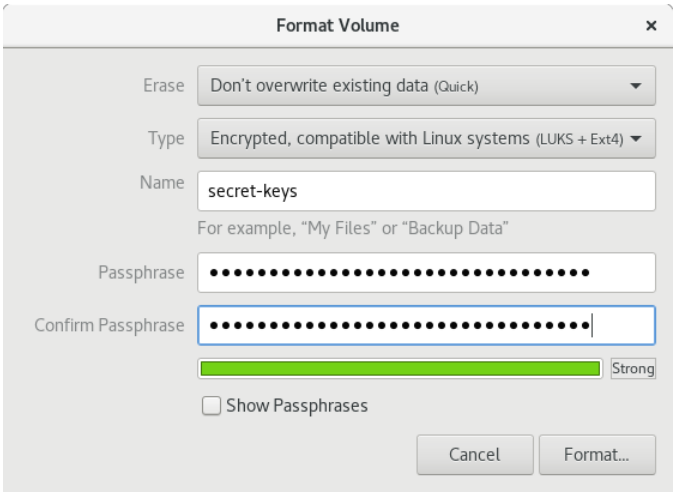

Figure 1.1: Formatting an encrypted partition in GNOME Disks.

The Sneaker Net File System

Unfortunately, most security tokens don't store the public key or have any storage space on them. Thus, to transfer the public key to the main computer, we need a third storage device.

This time, when partitioning the file system, name the file system `sneaker-net`, and use an unencrypted `ext4` file system. After it has been formatted, mount the partition. The file system will be mounted under `/media/amnesia/sneaker-net`.

1.3.6 Generating the Keys

We can finally generate the keys. At the beginning of this chapter, we generated a simple key that had a single subkey. When using a security token, we also want at least a signing subkey. If you plan to use your security token to authenticate `ssh` connections, then you'll also need an authentication subkey. The reason to have a signing subkey is to further isolate the powerful certification-capable key from your online devices: the certification key not only determines your key's identity, but it is also used to create new subkeys. Thus, using the master key not only as a certification key, but also a signing key would mean that if the security token is lost, then we'd have to create a new OpenPGP key. If this happens when using a separate

signing subkey, it is only necessary to revoke the signing subkey, and create a new one.

The following transcript shows how to create a certification-capable master key, and three subkeys. Note that we first change `gpg` 's home directory to be on the encrypted file system.

```
$ mkdir -p /media/amnesia/secret-keys/gnupg
$ export GNUPGHOME=/media/amnesia/secret-keys/gnupg
$ gpg --quick-gen-key 'Juliet Capulet <juliet@gnupg.net>' rsa cert 2y
gpg: WARNING: unsafe permissions on homedir '/media/amnesia/secret-keys
gpg: keybox '/media/amnesia/secret-keys/gnupg/pubring.kbx' created
gpg: /media/amnesia/secret-keys/gnupg/trustdb.gpg: trustdb created
gpg: key E9794A89BDB70380 marked as ultimately trusted
gpg: directory '/media/amnesia/secret-keys/gnupg/openpgp-revocs.d' crea
gpg: revocation certificate stored as '/media/amnesia/secret-keys/gnupg
public and secret key created and signed.

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.
pub   rsa2048 2017-08-14 [C] [expires: 2019-08-14]
      635D6A0EA043F835A1FFD9A7E9794A89BDB70380
uid                      Juliet Capulet <juliet@gnupg.net>

$ gpg --quick-addkey 635D6A0EA043F835A1FFD9A7E9794A89BDB70380 rsa encr
$ gpg --quick-addkey 635D6A0EA043F835A1FFD9A7E9794A89BDB70380 rsa sign
$ gpg --quick-addkey 635D6A0EA043F835A1FFD9A7E9794A89BDB70380 rsa auth
```

Listing the key, we can see that we got the desired structure:

```
$ gpg -K
gpg: WARNING: unsafe permissions on homedir '/media/amnesia/secret-keys
gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: next trustdb check due at 2019-08-14
/media/amnesia/secret-keys/gnupg/pubring.kbx
------------------------------------------
sec   rsa2048 2017-08-14 [C] [expires: 2019-08-14]
      635D6A0EA043F835A1FFD9A7E9794A89BDB70380
```

```
uid           [ultimate] Juliet Capulet <juliet@gnupg.net>
ssb   rsa2048 2017-08-14 [E] [expires: 2018-08-14]
ssb   rsa2048 2017-08-14 [S] [expires: 2018-08-14]
ssb   rsa2048 2017-08-14 [A] [expires: 2018-08-14]
```

Because the OpenPGP card doesn't include the public key, it is necessary to use the sneaker net storage device to transfer the public key from the offline machine to the online machine:

```
$ gpg -a --export 635D6A0EA043F835A1FFD9A7E9794A89BDB70380 > \
> /media/amnesia/sneaker-net/635D6A0EA043F835A1FFD9A7E9794A89BDB70380.pub
```

If your Tails machine is connected to the Internet, you could transfer the public key by uploading it to a key server or a website, and then retrieving it with the online machine.

1.3.7 Saving Your Progress

Just in case, we make a mistake, it is useful to create a snapshot of the secret key material:

```
$ gpg --export-secret-keys > /media/amnesia/secret-keys/secret-keys.gpg
```

To restore, we can import the key and mark it as ultimately trusted:

```
$ rm -rf /media/amnesia/secret-keys/gnupg
$ mkdir /media/amnesia/secret-keys/gnupg
$ gpg --import secret-keys.gpg
gpg: WARNING: unsafe permissions on homedir '/media/amnesia/secret-keys/gnupg'
gpg: keybox '/media/amnesia/secret-keys/gnupg/pubring.kbx' created
gpg: /media/amnesia/secret-keys/gnupg/trustdb.gpg: trustdb created
gpg: key E9794A89BDB70380: public key "Juliet Capulet <juliet@gnupg.net>" impo
gpg: key E9794A89BDB70380: secret key imported
gpg: Total number processed: 1
gpg:               imported: 1
gpg:       secret keys read: 1
gpg:   secret keys imported: 1
$ gpg --edit-key juliet@gnupg.net
...
```

```
Please decide how far you trust this user to correctly verify other use
(by looking at passports, checking fingerprints from different sources,

  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
```

1.3.8 Creating a Backup

The Revocation Certificate

It is almost certainly reasonable to store the revocation certificate on the main computer. Thus, we can copy it to our sneaker net device:

```
$ cp /media/amnesia/secret-keys/gnupg/openpgp-revocs.d/635D6A0EA043F835
> /media/amnesia/sneaker-net
```

Just remember to actually copy it to someplace that is regularly backed up.

The Secret Key

Backing up the secret key is more difficult, because its security requirements are much higher given the consequences of a compromise. Paperkey is a relatively convenient way to back up secret key material on paper. To do this, you have to to attach and configure a printer. Make sure the printer is not connected to the network, and reset the printer afterwards.

```
$ paperkey --secret-key /media/amnesia/secret-keys/secret-keys.gpg
# Secret portions of key 635D6A0EA043F835A1FFD9A7E9794A89BDB70380
# Base16 data extracted Mon Aug 14 13:36:43 2017
# Created with paperkey 1.3 by David Shaw
#
# File format:
```

```
# a) 1 octet:  Version of the paperkey format (currently 0).
# b) 1 octet:  OpenPGP key or subkey version (currently 4)
# c) n octets: Key fingerprint (20 octets for a version 4 key or subkey)
# d) 2 octets: 16-bit big endian length of the following secret data
# e) n octets: Secret data: a partial OpenPGP secret key or subkey packet as
#              specified in RFC 4880, starting with the string-to-key usage
#              octet and continuing until the end of the packet.
# Repeat fields b through e as needed to cover all subkeys.
#
# To recover a secret key without using the paperkey program, use the
# key fingerprint to match an existing public key packet with the
# corresponding secret data from the paper key.  Next, append this secret
# data to the public key packet.  Finally, switch the public key packet tag
# from 6 to 5 (14 to 7 for subkeys).  This will recreate the original secret
# key or secret subkey packet.  Repeat as needed for all public key or subkey
# packets in the public key.  All other packets (user IDs, signatures, etc.)
# may simply be copied from the public key.
#
# Each base16 line ends with a CRC-24 of that line.
# The entire block of data ends with a CRC-24 of the entire block of data.

  1: 00 04 63 5D 6A 0E A0 43 F8 35 A1 FF D9 A7 E9 79 4A 89 BD B7 03 80 1D7942
  2: 02 B9 FE 07 03 02 E7 84 42 CB 86 E4 73 CA DB 47 A2 C0 4F 0A BB 57 2C46C8
  3: 04 63 BF E6 11 52 C4 F3 7A BB 12 34 66 DB 79 5A 89 E1 C2 8D 2E 10 603062
  4: 0B 3D 57 0A FD ED 8A 97 71 0B 51 EB 31 C4 02 28 C1 6E 64 18 B9 2C 8470F7
...
132: C5CD3A
```

Note: when you restore the key, you'll still need the key's passphrase.

Another way to backup the secret key is to copy it to another encrypted storage device.

1.3.9 Copying the Keys to the Security Token

We can finally copy the keys to the security token. This is done using `gpg` 's `--card-edit` interface. To add keys to the card, you'll need to enter the admin PIN, not the user PIN.

The subcommand to move a key to a security token is `keytocard`. `keytocard` works on the currently active subkey. The `key` subcommand

is used to select or deselect a key. Since some operations can operate on multiple keys, it is necessary to explicitly deselect keys. Selected keys are shown with a $\star$.

```
$ gpg --edit-key 635D6A0EA043F835A1FFD9A7E9794A89BDB70380
Secret key is available.

sec  rsa2048/E9794A89BDB70380
     created: 2017-08-14  expires: 2019-08-14  usage: C
     trust: ultimate      validity: ultimate
ssb  rsa2048/F8D8ED7BB1A2A8F6
     created: 2017-08-14  expires: 2018-08-14  usage: E
ssb  rsa2048/305A846803A91753
     created: 2017-08-14  expires: 2018-08-14  usage: S
ssb  rsa2048/300BA8EE1B5EDEED
     created: 2017-08-14  expires: 2018-08-14  usage: A
[ultimate] (1). Juliet Capulet <juliet@gnupg.net>

gpg> key 1    # Select the first subkey.

sec  rsa2048/E9794A89BDB70380
     created: 2017-08-14  expires: 2019-08-14  usage: C
     trust: ultimate      validity: ultimate
ssb* rsa2048/F8D8ED7BB1A2A8F6
     created: 2017-08-14  expires: 2018-08-14  usage: E
ssb  rsa2048/305A846803A91753
     created: 2017-08-14  expires: 2018-08-14  usage: S
ssb  rsa2048/300BA8EE1B5EDEED
     created: 2017-08-14  expires: 2018-08-14  usage: A
[ultimate] (1). Juliet Capulet <juliet@gnupg.net>

gpg> keytocard
Please select where to store the key:
   (2) Encryption key
Your selection? 2
...
gpg> key 2    # Select the second subkey.
```

```
sec  rsa2048/E9794A89BDB70380
     created: 2017-08-14  expires: 2019-08-14  usage: C
     trust: ultimate      validity: ultimate
ssb* rsa2048/F8D8ED7BB1A2A8F6
     created: 2017-08-14  expires: 2018-08-14  usage: E
ssb* rsa2048/305A846803A91753
     created: 2017-08-14  expires: 2018-08-14  usage: S
ssb  rsa2048/300BA8EE1B5EDEED
     created: 2017-08-14  expires: 2018-08-14  usage: A
[ultimate] (1). Juliet Capulet <juliet@gnupg.net>

gpg> key 1    # Deselect the first subkey.

sec  rsa2048/E9794A89BDB70380
     created: 2017-08-14  expires: 2019-08-14  usage: C
     trust: ultimate      validity: ultimate
ssb  rsa2048/F8D8ED7BB1A2A8F6
     created: 2017-08-14  expires: 2018-08-14  usage: E
ssb* rsa2048/305A846803A91753
     created: 2017-08-14  expires: 2018-08-14  usage: S
ssb  rsa2048/300BA8EE1B5EDEED
     created: 2017-08-14  expires: 2018-08-14  usage: A
[ultimate] (1). Juliet Capulet <juliet@gnupg.net>

gpg> keytocard
Please select where to store the key:
   (1) Signature key
   (3) Authentication key
Your selection? 1
...
gpg> key 3    # Select the third subkey.
...
gpg> key 2    # Deselect the second subkey.
...
gpg> keytocard
Please select where to store the key:
   (3) Authentication key
Your selection? 3
```

```
...
gpg> quit
Save changes? (y/N) n
Quit without saving? (y/N) y
```

At the end of the transcript, we explicitly do *not* save the changes. Saving the changes would cause the secret key material for the corresponding subkey to be deleted. The reason for this is that the `keytocard` command doesn't copy, but *moves* the secret key material to the card. Saving without quitting inhibits this side effect. In practice, this isn't a problem if you backed up the secret key material, as recommended above.

Using `--card-status`, we can see that the keys were successfully loaded on to the security token:

```
$ gpg --card-status
Reader ...........: SCM Microsystems Inc. SPR 532 [Vendor Interface] (2
Application ID ...: D2760001240102010005000002D9D0000
Version ..........: 2.1
Manufacturer .....: ZeitControl
Serial number ....: 00002D9D
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 ...: rsa2048 rsa2048 rsa2048
Max. PIN lengths .: 32 32 32
PIN retry counter : 3 0 3
Signature counter : 0
Signature key ....: A17A D462 C473 51AD D0E8  988B 305A 8468 03A9 1753
      created ....: 2017-08-14 13:07:49
Encryption key....: C9CD 8F3D ECDE BB7E 720A  7CD9 F8D8 ED7B B1A2 A8F6
      created ....: 2017-08-14 13:07:32
Authentication key: 9308 3590 BCD9 3CC5 044C  BEAD 300B A8EE 1B5E DEED
      created ....: 2017-08-14 13:08:04
General key info..: sub  rsa2048/305A846803A91753 2017-08-14 Juliet Cap
sec   rsa2048/E9794A89BDB70380  created: 2017-08-14  expires: 2019-08-1
```

```
ssb   rsa2048/F8D8ED7BB1A2A8F6  created: 2017-08-14  expires: 2018-08-14
ssb   rsa2048/305A846803A91753  created: 2017-08-14  expires: 2018-08-14
ssb   rsa2048/300BA8EE1B5EDEED  created: 2017-08-14  expires: 2018-08-14
```

You can now shut Tails down.

1.3.10 Using the Keys

To actually use the keys on the security token, we need to do four more minor things.

First, plug the security token into your computer, and run `--card-status`. This command makes sure that `gpg` can actually see the card:

```
$ gpg --card-status
Reader ...........: SCM Microsystems Inc. SPR 532 [Vendor Interface] (21251019
Application ID ...: D2760001240102010005000002D9D0000
Version ..........: 2.1
Manufacturer .....: ZeitControl
Serial number ....: 00002D9D
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 ...: rsa2048 rsa2048 rsa2048
Max. PIN lengths .: 32 32 32
PIN retry counter : 3 0 3
Signature counter : 0
Signature key ....: A17A D462 C473 51AD D0E8  988B 305A 8468 03A9 1753
      created ....: 2017-08-14 13:07:49
Encryption key....: C9CD 8F3D ECDE BB7E 720A  7CD9 F8D8 ED7B B1A2 A8F6
      created ....: 2017-08-14 13:07:32
Authentication key: 9308 3590 BCD9 3CC5 044C  BEAD 300B A8EE 1B5E DEED
      created ....: 2017-08-14 13:08:04
General key info..: [none]
```

Although the security token is recognized (and the keys are loaded), you can't yet use the keys, because `gpg` is missing the public keys. Insert and mount the sneaker net device, and import them:

```
$ gpg --import /media/juliet/sneaker-net/635D6A0EA043F835A1FFD9A7E9794
gpg: Total number processed: 1
gpg:               imported: 1
```

Listing the secret key, we can see that it is now available.

```
$ gpg -K 635D6A0EA043F835A1FFD9A7E9794A89BDB70380
sec#  rsa2048/0xE9794A89BDB70380 2017-08-14 [C] [expires: 2019-08-14]
      Key fingerprint = 635D 6A0E A043 F835 A1FF  D9A7 E979 4A89 BDB7 0
uid                   [unknown] Juliet Capulet <juliet@gnupg.net>
ssb>  rsa2048/0xF8D8ED7BB1A2A8F6 2017-08-14 [E] [expires: 2018-08-14]
ssb>  rsa2048/0x305A846803A91753 2017-08-14 [S] [expires: 2018-08-14]
ssb>  rsa2048/0x300BA8EE1B5EDEED 2017-08-14 [A] [expires: 2018-08-14]
```

The `#` after the `sec` header for the master key means that the master key is not available; the `>` next to the `ssb` keys means that the keys are on a security token.

Looking at the above output, we also see that the key is not marked as trusted. In order for certifications by this key to be respected, it is necessary to mark the key as ultimately trusted. This can be done using the `--edit-key` interface.

```
$ gpg --edit-key 635D6A0EA043F835A1FFD9A7E9794A89BDB70380
Secret key is available.
...
gpg> trust
...
Please decide how far you trust this user to correctly verify other use
(by looking at passports, checking fingerprints from different sources,

  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
...
```

Finally, you'll also want to publish your key so that others can more easily find it.

```
$ gpg --send-key 635D6A0EA043F835A1FFD9A7E9794A89BDB70380
```

1.3.11 Saving the Revocation Certificate

Before unmounting the device, there is one last thing that we need to do: we need to copy the revocation certificate someplace that is backed up.

```
$ gpg --import /media/juliet/sneaker-net/635D6A0EA043F835A1FFD9A7E9794A89BDB70
> ~/.gnupg
gpg: Total number processed: 1
gpg:               imported: 1
```

1.3.12 Signing Keys with an Offline Master

When it comes to signing keys, having an offline master key is a pain: it is necessary to download the public keys on a network-connected computer, transfer them to a removable storage device, sign them on the offline computer, and then move the signatures back to the main computer. The key signing tool `caff` partially automates this workflow, but it is still inconvenient.

For most users, it would be nice to somehow separate the "modify the key" and "certify other keys" capabilities: only the former capability is highly sensitive.

With a few small tricks, this is possible. The basic idea is to create a secondary, certification-only key, which is not offline, and to have the offline key designate it as a trusted introducer using a so-called *trust signature*. Then, you sign other people's keys using the second key, and ask people sign your main key. If anyone sets your main key as a trusted introducer, then they will automatically trust your secondary key by way of the trust signature. If the second key is somehow compromised, it can be revoked, and replaced with a new key. And, any signatures can be recreated with the new key.

To create a certification-only key, do the following:

```
$ gpg --quick-gen-key 'Juliet Capulet (certification key)' rsa cert 2y
gpg: key 0xCD6AF594BAA8EF38 marked as ultimately trusted
```

```
gpg: revocation certificate stored as '/home/us/.gnupg/openpgp-revocs.
public and secret key created and signed.

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.
pub   rsa2048/0xCD6AF594BAA8EF38 2017-08-14 [C] [expires: 2019-08-14]
      Key fingerprint = 149D 0735 A25E 63B1 EC9F  EEBD CD6A F594 BAA8 E
uid                              Juliet Capulet (certification key)
```

There are two things to note about this key. First, the user ID doesn't include an email address. This is useful to prevent people who search the key servers by email address from finding the wrong key. Although searching a key server by email address is strongly discouraged, there is no need to make such users' lives worse than necessary. Second, the key only includes a certification-capable key; there are no signing or encryption subkeys. This prevents people from accidentally encrypting data to your secondary key, or a misconfigured GnuPG from accidentally using it to generate a signature.

After creating the key, we need to sign it using the main key. This requires transferring the public key to the offline computer.

```
$ gpg --export 149D0735A25E63B1EC9FEEBDCD6AF594BAA8EF38 > \
> /media/juliet/sneaker-net/149D0735A25E63B1EC9FEEBDCD6AF594BAA8EF38.pu
```

Then running the following on the offline computer after remounting the encrypted file system with the secret key, and the sneaker net file system:

```
$ export GNUPGHOME=/media/amnesia/secret-keys/gnupg
$ gpg --import /media/amnesia/sneaker-net/149D0735A25E63B1EC9FEEBDCD6AF
gpg: key CD6AF594BAA8EF38: public key "Juliet Capulet (certification ke
gpg: Total number processed: 1
gpg:               imported: 1
$ gpg --edit-key 149D0735A25E63B1EC9FEEBDCD6AF594BAA8EF38
pub  rsa2048/CD6AF594BAA8EF38
     created: 2017-08-14  expires: 2019-08-14  usage: C
     trust: unknown       validity: unknown
[ unknown] (1). Juliet Capulet (certification key)
```

```
gpg> tsign

pub  rsa2048/CD6AF594BAA8EF38
     created: 2017-08-14  expires: 2019-08-14  usage: C
     trust: unknown       validity: unknown
 Primary key fingerprint: 149D 0735 A25E 63B1 EC9F  EEBD CD6A F594 BAA8 EF38

     Juliet Capulet (certification key)

This key is due to expire on 2019-08-14.
Please decide how far you trust this user to correctly verify other users' key
(by looking at passports, checking fingerprints from different sources, etc.)

  1 = I trust marginally
  2 = I trust fully

Your selection? 2

Please enter the depth of this trust signature.
A depth greater than 1 allows the key you are signing to make
trust signatures on your behalf.

Your selection? 2

Please enter a domain to restrict this signature, or enter for none.

Your selection?

Are you sure that you want to sign this key with your
key "Juliet Capulet <juliet@gnupg.net>" (E9794A89BDB70380)

Really sign? (y/N) y

gpg> Save changes? (y/N) y
$ gpg --export-options backup \
> --export 149D0735A25E63B1EC9FEEBDCD6AF594BAA8EF38 > \
> /media/amnesia/sneaker-net/149D0735A25E63B1EC9FEEBDCD6AF594BAA8EF38.pub
```

Using a value of `2` for the depth parameter to the `tsign` subcommand means that anyone who considers the main key to be a trusted introducer will also consider the certification key to be a trusted introducer. (Using a value of `1` is equivalent to a normal signature, which simply verifies the target, but does not claim that the key's own should be treated as a trusted introducer.)

It is possible to restrict the domains over which the trusted signature is valid. But, support for this in GnuPG is incomplete. For instance, this feature is currently not supported on Windows.

After creating the signature, move the signed public key back to the main computer, import it, and publish it.

```
$ gpg --import /media/juliet/sneaker-net/149D0735A25E63B1EC9FEEBDCD6AF5
gpg: key 0xCD6AF594BAA8EF38: "Juliet Capulet (certification key)" 1 new
gpg: WARNING: server 'gpg-agent' is older than us (2.1.18 < 2.1.23-beta
gpg: Note: Outdated servers may lack important security fixes.
gpg: Note: Use the command "gpgconf --kill all" to restart them.
gpg: Total number processed: 1
gpg:         new signatures: 1
# gpg --send-key 149D0735A25E63B1EC9FEEBDCD6AF594BAA8EF38
```

It is also a good idea to have the certification key cross sign the main key. (The `-u` option indicates what key to use for the signature. This is useful if you have multiple keys.)

```
$ gpg -u 149D0735A25E63B1EC9FEEBDCD6AF594BAA8EF38 \
> --edit-key 635D6A0EA043F835A1FFD9A7E9794A89BDB70380
Secret key is available.

pub  rsa2048/0xE9794A89BDB70380
     created: 2017-08-14  expires: 2019-08-14  usage: C
     trust: ultimate      validity: ultimate
ssb  rsa2048/0xF8D8ED7BB1A2A8F6
     created: 2017-08-14  expires: 2018-08-14  usage: E
     card-no: 0005 00002D9D
ssb  rsa2048/0x305A846803A91753
     created: 2017-08-14  expires: 2018-08-14  usage: S
     card-no: 0005 00002D9D
ssb  rsa2048/0x300BA8EE1B5EDEED
```

```
     created: 2017-08-14  expires: 2018-08-14  usage: A
     card-no: 0005 00002D9D
[ultimate] (1). Juliet Capulet <juliet@gnupg.net>

gpg> sign

pub  rsa2048/0xE9794A89BDB70380
     created: 2017-08-14  expires: 2019-08-14  usage: C
     trust: ultimate      validity: ultimate
 Primary key fingerprint: 635D 6A0E A043 F835 A1FF  D9A7 E979 4A89 BDB7 0380

     Juliet Capulet <juliet@gnupg.net>

This key is due to expire on 2019-08-14.
Are you sure that you want to sign this key with your
key "Juliet Capulet (certification key)" (0xCD6AF594BAA8EF38)

Really sign? (y/N) y

gpg> Save changes? (y/N) y
```

1.4 Key Expiration

OpenPGP includes a key expiry mechanism. An expired key is like a revoked key: `gpg` won't encrypt a message to it or rely on it. But, unlike a revocation certificate, the expiry information is published immediately. Thus, if the user loses access to the key *and* the revocation certificate, the key will still eventually be considered invalid. This is particularly useful for users who uninstall GnuPG without first publishing a revocation certificate. Another difference is that whereas revocation certificates can't be rescinded once they are published, it is possible to change when a key expires. This is important as keys are intended to be long-term identities, but the time until a key expires should be relatively short.

As of GnuPG 2.1, GnuPG automatically sets newly created primary keys to expire in two years. The easiest way to change the expiration time is to use the `--quick-set-expire` command. The following command sets the primary key to expire in two years relative to when the command

was issued:

```
$ gpg --quick-set-expire 149D0735A25E63B1EC9FEEBDCD6AF594BAA8EF38 2y
```

If the expiration of a subkey needs to be extended, this can be done as follows:

```
$ gpg --quick-set-expire 149D0735A25E63B1EC9FEEBDCD6AF594BAA8EF38 2y \
> C9CD8F3DECDEBB7E720A7CD9F8D8ED7BB1A2A8F6
```

Unfortunately, using `--quick-set-expire` to extend a subkey's expiration is only supported since 2.1.22, which was released in July 2017. As such, this functionality is not supported by the version of GnuPG shipped with Debian 9 (stretch). In this case, it is necessary to use the `expire` command from the `--edit-key` interface to extend a subkey's expiration.

After extending a key's expiration, don't forget to publish the updated key (e.g., using `--send-key`) so that your communication partners see the change. If the expiration is adjusted on an offline computer, you'll need to first export the updated key (`--export`) to a removable storage device, and then import it on an online computer.

To better avoid the problems associated with an expired key, it is better to extend the expiration date two or three months before the expiry so that any automatic update mechanism picks up the change, before the user sees an error.

1.5 Subkey Rotation

A user can approximate forward secrecy by regularly rotating her encryption and signing subkeys [6]. Unfortunately, since endpoint security—not the cryptography—tends to be the weak point in the system, this only really makes sense for people who store their keys on a security token. Unfortunately, if you are using a security token, then you won't be able to store both your old keys and your new keys on the same token: the OpenPGP card specification only supports a single encryption key. Although it is possible to carry multiple security tokens, this is often inconvenient. In particular, it becomes unmanageable after a few key rotations. Thus, in practice, this only makes sense if you are willing to forego access to old encrypted data, which is not how most people use OpenPGP.

Rotating keys is as simple as revoking the old keys, and generating new subkeys. To revoke a subkey, it is necessary to use the `--edit-key` interface, select the subkeys to revoke using the `key` subcommand, and use the `revkey` subcommand to revoke the selected subkeys. This is illustrated below:

```
$ gpg --edit-key 635D6A0EA043F835A1FFD9A7E9794A89BDB70380
Secret key is available.

sec  rsa2048/E9794A89BDB70380
     created: 2017-08-14  expires: 2019-08-14  usage: C
     trust: ultimate      validity: ultimate
ssb  rsa2048/F8D8ED7BB1A2A8F6
     created: 2017-08-14  expires: 2018-08-14  usage: E
ssb  rsa2048/305A846803A91753
     created: 2017-08-14  expires: 2018-08-14  usage: S
ssb  rsa2048/300BA8EE1B5EDEED
     created: 2017-08-14  expires: 2018-08-14  usage: A
[ultimate] (1). Juliet Capulet <juliet@gnupg.net>

gpg> key *   # Select all keys.  You can also use key N
             # to select the Nth subkey.

sec  rsa2048/E9794A89BDB70380
     created: 2017-08-14  expires: 2019-08-14  usage: C
     trust: ultimate      validity: ultimate
ssb* rsa2048/F8D8ED7BB1A2A8F6
     created: 2017-08-14  expires: 2018-08-14  usage: E
ssb* rsa2048/305A846803A91753
     created: 2017-08-14  expires: 2018-08-14  usage: S
ssb* rsa2048/300BA8EE1B5EDEED
     created: 2017-08-14  expires: 2018-08-14  usage: A
[ultimate] (1). Juliet Capulet <juliet@gnupg.net>

gpg> revkey
Do you really want to revoke the selected subkeys? (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
Your decision? 2
Enter an optional description; end it with an empty line:
>
Reason for revocation: Key is superseded
(No description given)
Is this okay? (y/N) y

sec  rsa2048/E9794A89BDB70380
     created: 2017-08-14  expires: 2019-08-14  usage: C
     trust: ultimate      validity: ultimate
The following key was revoked on 2017-08-15 by RSA key E9794A89BDB7038
ssb  rsa2048/F8D8ED7BB1A2A8F6
     created: 2017-08-14  revoked: 2017-08-15  usage: E
The following key was revoked on 2017-08-15 by RSA key E9794A89BDB7038
ssb  rsa2048/305A846803A91753
     created: 2017-08-14  revoked: 2017-08-15  usage: S
The following key was revoked on 2017-08-15 by RSA key E9794A89BDB7038
ssb  rsa2048/300BA8EE1B5EDEED
     created: 2017-08-14  revoked: 2017-08-15  usage: A
[ultimate] (1). Juliet Capulet <juliet@gnupg.net>
```

Bibliography

[1] Bruce Schneier. NSA surveillance: A guide to staying secure. https://www.theguardian.com/world/2013/sep/05/nsa-how-to-remain-secure-surveillance, September 2013.

[2] Achim Pietig. Functional specification of the openpgp application on iso smart card operating systems. https://gnupg.org/ftp/specs/OpenPGP-smart-card-application-3.3.pdf, June 2017.

[3] Jakob Ehrensvärd. Secure hardware vs. open source. https://www.yubico.com/2016/05/secure-hardware-vs-open-source/, May 2016. Last accessed: August 2, 2017.

[4] Wikipedia. Dual ec drbg — wikipedia, the free encyclopedia, 2017. [Online; accessed 2-August-2017].

[5] J Alex Halderman, Seth D Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A Calandrino, Ariel J Feldman, Jacob Appelbaum, and Edward W Felten. Lest we remember: cold-boot attacks on encryption keys. *Communications of the ACM*, 52(5):91–98, 2009.

[6] Ian Brown, Adam Back, and Ben Laurie. Forward Secrecy Extensions for OpenPGP. Internet-Draft draft-brown-pgp-pfs-03, IETF Secretariat, October 2001. https://tools.ietf.org/html/draft-brown-pgp-pfs-03.