Systems, Tools, and Terminal Science

Posted on

With our private and public key generated and stored, we can start using a few of GnuPG's features to sign, verify, encrypt, and decrypt files and messages for distribution over untrusted channels like the internet.

We'll start by signing a simple text file, using the `--clearsign` option. This includes the signature in the message, which we can then distribute to people to read. Here's the contents of `message.txt`:

```
This is a public message from Tom Ryder.
```

We'll sign that with our new private key like so:

```
$ gpg --clearsign message.txt
```

We're prompted for our passphrase for the private key:

```
You need a passphrase to unlock the secret key for
user: "Tom Ryder (Test Key Only) <tom@sanctum.geek.nz>"
4096-bit RSA key, ID 040FE79B, created 2013-03-23
```

Having provided that, the file `message.txt.asc` is created, with PGP sections and a plaintext ASCII signature:

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA256

This is a public message from Tom Ryder.
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.10 (GNU/Linux)

iQIcBAEBCAAGBQJRTQcTAAoJEGIHQ4oED+ebtjoP/19PlndkGhR46BA6YZmDVdC1
Snk9aXe4Eo42kRpW13bjo8xg+pb+U26ylHkH52OBB7fJ3/BR5eZMug/RXJLGIf+U
aiylbGVz4dGjkeTCSxtg1TvcyEtzhm1ETfIWLarboj9PPBJfO1QIh4uPDkD7kb6+
+OnpdxPURbiJ03osu2Mj2fFq5wYT0La+I9BdKUAJmS8zt+CaTirKw+xF6l0sEAVv
lqslWjEwF9JCfumKAj8aMBeZMndoKRqW18ZHoYJdP0x3g1SUKAjZ/NRRUGug6Eg+
JTJ82ETCRKGKmYFLkHJ6iCaucrmhLTd9IYyEQZE/weUuClKqtshollhNHF1dD3SF
fWMPq0+29DInjlXwIkXyzVDln1wULNzbd5zv5Wg5b6lSCZlwH0xCrNjiYO7f413c
Ty4q6SqtRFUommnpMA5XcmX7ebbUpMfqfqqzoLqeTpA15Yuhh3DDR6NoMN82oLyF
FFt7UZh/JlYMc8GOnEqyZfT7d57FbKSLn3vpZbH9QXNFWG6/oZabFFyRm8r7k8F2
FzYTdyp59O0dW4T5OJ6a/xo/OnZutUN1RqW6ZJS19Xb4/5eEohFAFL9cDVLuC6zo
HmlA2m5zwv8aYbJad6Rk6+vpQAAdxHgNq/VYdcOfLOtcJAE6Jnm3alVyeXb1PbMB
WBxaM998Z6R8VmeMB7gQ
=WMzO
-----END PGP SIGNATURE-----
```

Note that the message itself is plainly readable; this message isn't encrypted, it's just verified as having been written by a particular person, and not altered since it was written.

Now anyone who has our public key on their keyring (as we ourselves do) can verify that it was actually us who wrote this message:

```
$ gpg --verify message.txt.asc
gpg: Signature made Sat 23 Mar 2013 14:32:17 NZDT using RSA key ID 040
gpg: Good signature from "Tom Ryder (Test Key Only) <tom@sanctum.geek.
```

If anybody tampers with the message, even something like removing a period from the end of a sentence, the verification will fail, suggesting the message was tampered with:

```
$ gpg --verify message.txt.asc
gpg: Signature made Sat 23 Mar 2013 14:32:17 NZDT using RSA key ID 040
gpg: BAD signature from "Tom Ryder (Test Key Only) <tom@sanctum.geek.n
```

For all other files, we likely need to make the signature file separate with a **detached signature**:

```
$ gpg --armor --detach-sign archive.tar.gz
```

This produces a file `archive.tar.gz.asc` in the same directory, containing the signature. We use `--armor` to make the signature in ASCII, which makes for a longer file but easier distribution online.

In this case, both the file and the signature are required for verification; put the signature file first when you check this:

```
$ gpg --verify archive.tar.gz.asc archive.tar.gz
```

You can use this method to verify software downloads from trusted sources, such as the . First, we would download and import all their public keys at the URL they nominate:

```
$ wget http://www.apache.org/dist/httpd/KEYS
$ gpg --import KEYS
```

We could then download an Apache HTTPD release, along with its key, from an arbitrary mirror:

```
$ wget http://www.example.com/apache/httpd/httpd-2.4.4.tar.gz
$ wget https://www.apache.org/dist/httpd/httpd-2.4.4.tar.gz.asc
```

We can then use the key and signature to verify that it's an uncompromised copy of the original file signed by the developers:

```
$ gpg --verify httpd-2.4.4.tar.gz.asc httpd-2.4.4.tar.gz
gpg: Signature made Tue 19 Feb 2013 09:28:39 NZDT using RSA key ID 7914
gpg: Good signature from "Jim Jagielski (Release Signing Key) <jim@apa
gpg:                 aka "Jim Jagielski <jim@jaguNET.com>"
gpg:                 aka "Jim Jagielski <jim@jimjag.com>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the
Primary key fingerprint: A93D 62EC C3C8 EA12 DB22  0EC9 34EA 76E6 7914
```

Note that the `gpg` output cautions that this is still not perfect assurance that the release actually came from Jim Jagielski, because we've never met him and can't absolutely, definitely say that this is his public key. , we can however see a lot of other Apache developers have signed his key, which looks promising, but do we know who *they* are?

Despite the lack of absolute certainty, when downloading from mirrors this is a lot better (and harder to exploit) than simply downloading without validating or checksumming at all, given that the signature and the `KEYS` file were downloaded from Apache's own site.

You will need to decide for yourself whether a person's public key really corresponds to them. This might extend to the point of arranging to meet them with government-issued identification!

We can encrypt a file so that only nominated people can decrypt and read it. In this case, we encrypt it not with our own private key, but with the recipient's public key. This means that they will be able to decrypt it using their own private key.

Here's the contents of `secret-message.txt`:

```
This is a secret message from Tom Ryder.
```

Now we need at least one recipient. Let's say this message was intended for my friend John Public. He's given me his public key in a file called `john-public.asc` on a USB drive in person; he even brought along his birth certificate and driver's license (which is weird, because I've known him since I was four).

To start with, I'll import his key into my keychain:

```
$ gpg --import john-public.asc
gpg: key 695195A5: public key "John Public (Main key) <johnpublic@exam
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
```

Now I can encrypt the message for only John to read. I like to use the 8-digit hex code for the key for `--recipient`, to make sure I've got the right person. You can see it in the output above, or in the output of `gpg --list-keys`.

```
$ gpg --armor --recipient 695195A5 --encrypt secret-message.txt
```

The encrypted message is written to `secret-message.txt.asc`:

```
-----BEGIN PGP MESSAGE-----
Version: GnuPG v1.4.10 (GNU/Linux)

hQEMAxiBb8eWSupuAQgAgOUQvqbTh60N6RQhDtP/bY9l+gjm4Grx5XcuhgQqK6pn
YtyPTKcpHdPK679lhbv0vE0RYe7pL+nBOngU1hCQYuGbRDZDxIXTIZW/rBvXbtHA
jgeSxrquad2totfh2nc7upePVCqXncPrLraJyDJBLLMrBHVvmOZymDabJbemOFuq
A/NbcmT3+osptvaEPFdlbgAW+J3vGxXMuQQYkT8GSnuutfEhZRb7SEL1ktaXwaMc
AA6NAan5ak7nCyDDHhDSDFMS9SQQHd8TDvQPF6OzRXlq26EOFD8HvlbDcgc51lbS
+N5nWaHM/CiuPh9dIOEV0H4Y8WDBdgkxp6kXKQfqb9JzAdwQ047r82SJAA7MSqCS
HRVtCRf5SNM12HqTRzF9XXum4uG+HXT6Bpy+K/lYpLgmHcHoUVKh8c2OcGaCHWQh
UC9B+aaThKdkxUfD/9tVIRmugjutgj7KdtDTGm+qLeCoJqp6HK5z5SX8Ha+P6/P5
hxinyw==
=kqUG
-----END PGP MESSAGE-----
```

Note that even I can’t read it, because I didn’t list myself as a recipient, and I don’t have access to John’s private key:

```
tom@tombox:~$ gpg --decrypt secret-message.txt.asc
gpg: encrypted with 2048-bit RSA key, ID 964AEA6E, created 2013-03-10
    "John Public (Main key) <johnpublic@example.com>"
gpg: decryption failed: secret key not available
```

However, on John’s computer, using his private key, he can decrypt and read it:

```
john@johnbox:~$ gpg --decrypt secret-message.txt.asc
gpg: encrypted with 2048-bit RSA key, ID 964AEA6E, created 2013-03-10
    "John Public (Main key) <johnpublic@example.com>"
This is a private, secret message from Tom Ryder.
```

If I wanted to make sure I could read the message too, I'd add my own public key to identify myself as a recipient when I encrypt it. Then either of us will be able to read it with our private keys (independently of the other):

```
$ gpg --recipient 695195A5 --recipient 040FE79B \
    --armor --encrypt secret-message.txt
```

Just to be thorough, we can sign the message as well to prove it came from us:

```
$ gpg --recipient 695195A5 --recipient 040FE79B \
    --armor --sign --encrypt secret-message.txt
```

Then when John runs the `--decrypt`, `gpg` will automatically verify the signature for us too, provided he has my public key in his keyring:

```
$ gpg --decrypt secret-message.txt.asc
gpg: encrypted with 2048-bit RSA key, ID 964AEA6E, created 2013-03-10
    "John Public (Main key) <johnpublic@example.com>"
gpg: encrypted with 4096-bit RSA key, ID AA159E5B, created 2013-03-23
    "Tom Ryder (Test Key Only) <tom@sanctum.geek.nz>"
This is a private, secret message from Tom Ryder.
gpg: Signature made Sat 23 Mar 2013 17:23:20 NZDT using RSA key ID 040
gpg: Good signature from "Tom Ryder (Test Key Only) <tom@sanctum.geek.
```

These are all the basic functions of GnuPG that will be useful to most people. We haven't considered here , or participating in the ; you should only look into this once you're happy with how your key setup is working, and are ready to publish your key for public use.

This entry is part 3 of 10 in the series .

This entry was posted in and tagged , , , , , , , , , , , by . Bookmark the .