GPG

One of the requirements for publishing your artifacts to the Central Repository, is that they have been signed with PGP. GnuPG or GPG is a freely available implementation of the OpenPGP standard. GPG provides you with the capability to generate a signature, manage keys, and verify signatures. This page documents usage of GPG as it relates to the Central Repository. In a nutshell you will have to

- create your own key pair
- and distribute it to a key server so that users can validate it

Installing GnuPG

Download the binary of GnuPG from https//www.gnupg.org/download/ or install it with your favorite package manager and verify it by running a gpg command with the `--version` flag.

```
$ gpg --version
gpg (GnuPG) 2.2.19
libgcrypt 1.8.5
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Home: /home/mylocaluser/.gnupg
Supported algorithms:
Pubkey: RSA, ELG, DSA, ECDH, ECDSA, EDDSA
Cipher: IDEA, 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH,
        CAMELLIA128, CAMELLIA192, CAMELLIA256
Hash: SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224
Compression: Uncompressed, ZIP, ZLIB, BZIP2
```

Generating a Key Pair

A key pair allows you to sign artifacts with GPG and users can subsequently validate that artifacts have been signed by you. You can generate a key with:

```
gpg --gen-key
```

Enter your name and email when asked for it and also, the time of validity for the key defaults to 2 years. Once they key is expired you can extend it, provided you own the key and therefore know the passphrase.

```
$ gpg --gen-key
gpg (GnuPG) 2.2.19; Copyright (C) 2019 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.

Note: Use "gpg --full-generate-key" for a full featured key generation dialog.

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

Real name: Central Repo Test
Email address: central@example.com
You selected this USER-ID:
    "Central Repo Test <central@example.com>"

Change (N)ame, (E)mail, or (O)kay/(Q)uit? O
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.
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 8190C4130ABA0F98 marked as ultimately trusted
gpg: revocation certificate stored as
'/home/mylocaluser/.gnupg/openpgp-revocs.d
/CA925CD6C9E8D064FF05B4728190C4130ABA0F98.rev'
public and secret key created and signed.

pub   rsa3072 2021-06-23 [SC] [expires: 2023-06-23]
      CA925CD6C9E8D064FF05B4728190C4130ABA0F98
uid                      Central Repo Test <central@example.com>
sub   rsa3072 2021-06-23 [E] [expires: 2023-06-23]
```

You have to provide your name and email. These identifiers are essential as they will be seen by anyone downloading a software artifact and validating a signature. Finally, you can provide a passphrase to protect your secret key. It is essential that you choose a secure passphrase and that you do not divulge it to any one. **This passphrase and your private key are all that is needed to sign artifacts with your signature.**

Listing Keys

Once key pair is generated, we can list them along with any other keys installed:

```
$ gpg --list-keys
/home/mylocaluser/.gnupg/pubring.kbx
---------------------------------
pub   rsa3072 2021-06-23 [SC] [expires: 2023-06-23]
      CA925CD6C9E8D064FF05B4728190C4130ABA0F98
uid           [ultimate] Central Repo Test <central@example.com>
sub   rsa3072 2021-06-23 [E] [expires: 2023-06-23]
```

The output displays the path to the public keyring file. The line starting with *pub* shows the size (rsa3072), the **keyid** (`CA925CD6C9E8D064FF05B4728190C4130ABA0F98`), and the creation date (2023-06-23) of the public key. Some values may vary depending on your GnuPG version, but you will definitely see the keyid or part of it (called shortID, last 8 characters of the keyid, in this example `0ABA0F98`) on the information listed.

The next line shows the UID of the key, which is composed of a name, a comment, and an email.

Hint

In case you have multiple keys, the local gpg will use the first listed signature key (`gpg --list-signatures`) for any publishing steps, if you need to use a specific key you could add the details of the gpg key inside a `<configuration>` section in your `pom.xml` and use local `settings.xml` to discover the passphrase via the signature keyname. You may need to use the signature keyid in **hexadecimal format**:

```
$ gpg --list-signatures --keyid-format 0xshort
/home/mylocaluser/.gnupg/pubring.kbx
---------------------------------
pub   rsa3072/0x3ABDEC12 2021-01-27 [SC] [expires: 2023-01-27]
      74524542545300A398653AB5242798823ABDEC12
uid           [ultimate] Other Name <otheremail@example.com>
sig 3        0x3ABDEC12 2021-01-27  Other Name <alarconj@gmail.com>
sub   rsa3072 2021-01-27 [E] [expires: 2023-01-27]
sig          0x3ABDEC12 2021-01-27  Julian Alarcon <alarconj@gmail.com>

pub   rsa3072/0x0ABA0F98 2021-06-23 [SC] [expires: 2022-03-21]
      CA925CD6C9E8D064FF05B4728190C4130ABA0F98
uid           [ultimate] Central Repo Test <central@example.com>
sig 3        0x0ABA0F98 2021-06-24  Central Repo Test <central@example.com>
sub   rsa3072/0x7C17C93B 2021-06-23 [E] [expires: 2023-06-23]
sig          0x0ABA0F98 2021-06-23  Central Repo Test <central@example.com>
```

You will find in the line that starts with **sig 3** that `0x3ABDEC12` is the signature keyid that you can use in your `pom.xml`.

Signing a File

To create an ASCII formatted signature for any file, run the following gpg command:

```
gpg -ab myfile.java
```

The `-a` option tells gpg to create ASCII armored output, the `-b` option tells gpg to make a detached signature. If your private key has a passphrase, you will be asked for it. Without a passphrase, all someone needs to forge an artifact signature is your private key. The passphrase is an extra level of recommended protection.

GPG will create a file like `temp.java.asc`, which is the signature of `myfile.java`.

You will want to distribute it along with the main file so the other can verify the main file using your public key:

```
$ gpg --verify myfile.java.asc
gpg: assuming signed data in 'myfile.java'
gpg: Signature made mié 23 jun 2021 16:06:54 -05
gpg:                using RSA key CA925CD6C9E8D064FF05B4728190C4130ABA0F98
gpg: Good signature from "Central Repo Test <central@example.com>" [ultimate]
```

Distributing Your Public Key

Since other people need your public key to verify your files, you have to distribute your public key to a key server:

```
gpg --keyserver keyserver.ubuntu.com --send-keys
CA925CD6C9E8D064FF05B4728190C4130ABA0F98
```

Important

As SKS Keyserver Network is being deprecated we recommend the use an specific GPG keyserver. Current GPG Keyservers supported by Central Servers are:

- `keyserver.ubuntu.com`
- `keys.openpgp.org`
- `pgp.mit.edu`

The `--keyserver` parameter identifies the target key server address and use `--send-keys` is the keyid of the key you want to distribute. You can get your keyid by listing the public keys.

Now other people can import your public key from the key server to their local machines:

```
gpg --keyserver keyserver.ubuntu.com --recv-keys
CA925CD6C9E8D064FF05B4728190C4130ABA0F98
```

Using Build Tools for Signing

Now that you have created and distributed your key, you can proceed to get the components automatically signed as part of your build. Depending on your build tool this setup will be different and you can find out more about these next steps in the specific sections:

- Apache Maven
- Apache Ant
- Gradle
- sbt
- Manual Staging Bundle Creation and Deployment

Dealing with Expired Keys

When you generate your PGP key, you need to specify how long the key should be valid. After that period you can edit your existing key to extend it's valid time.

For example, this is key pair which expires on `2023-06-23`:

```
$ gpg --list-keys
/Users/mylocaluser/.gnupg/pubring.gpg
-------------------------------
pub   rsa3072 2021-06-23 [SC] [expires: 2023-06-23]
      CA925CD6C9E8D064FF05B4728190C4130ABA0F98
uid           [ultimate] Central Repo Test <central@example.com>
sub   rsa3072 2021-06-23 [E] [expires: 2023-06-23]
```

You can edit a key with the following command using the key id as parameter:

```
$ gpg --edit-key CA925CD6C9E8D064FF05B4728190C4130ABA0F98
gpg (GnuPG) 2.2.19; Copyright (C) 2019 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.

sec  rsa3072/8190C4130ABA0F98
     created: 2021-06-23  expires: 2023-06-23  usage: SC
     trust: ultimate      validity: ultimate
ssb  rsa3072/4656B4857C17C93B
     created: 2021-06-23  expires: 2023-06-23  usage: E
[ultimate] (1). Central Repo Test <central@example.com>

gpg>
```

There is only one key to edit, so I choose *1*:

```
gpg> 1

sec  rsa3072/8190C4130ABA0F98
     created: 2021-06-23  expires: 2023-06-23  usage: SC
     trust: ultimate      validity: ultimate
ssb  rsa3072/4656B4857C17C93B
     created: 2021-06-23  expires: 2023-06-23  usage: E
[ultimate] (1)* Central Repo Test <central@example.com>
```

You will see the `*` after (1), which mean you've selected this key to edit. To edit the keys expiration time, enter the following command:

```
gpg> expire
Changing expiration time for the primary key.
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)
```

Enter what you need, for example `9m` (9 months), and confirm it. You will be asked for the passphrase of that key. The last step about editing is saving it what you've done:

```
gpg> save
```

Now you can see your key's expires time is updated:

```
$ gpg --list-keys
/home/mylocaluser/.gnupg/pubring.kbx
---------------------------------
pub   rsa3072 2021-06-23 [SC] [expires: 2022-03-21]
      CA925CD6C9E8D064FF05B4728190C4130ABA0F98
uid           [ultimate] Central Repo Test <central@example.com>
```

```
sub   rsa3072 2021-06-23 [E] [expires: 2023-06-23]
```

Finally, distribute your public key again:

```
gpg --keyserver keyserver.ubuntu.com --send-keys
CA925CD6C9E8D064FF05B4728190C4130ABA0F98
```

Delete a Sub Key

Some PGP tools generates sub keys and use them for signing by default, but to make Maven tools recognize the signature, you must use the primary key to sign your artifacts.

This is a problem if you use it to sign artifacts and deploy artifacts to the Central Repository, because Maven as well as Nexus Repository Manager can only verify against a primary key.

To fix this problem you have to delete the sub signing key so PGP will use the primary key for signing. To get an idea whether you have a sub signing key, run command below with your own key ID:

```
$ gpg --edit-key CA925CD6C9E8D064FF05B4728190C4130ABA0F98
gpg (GnuPG) 2.2.19; Copyright (C) 2019 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.

sec  rsa3072/8190C4130ABA0F98
     created: 2021-06-23  expires: 2022-03-21  usage: SC
     trust: ultimate      validity: ultimate
ssb  rsa3072/4656B4857C17C93B
     created: 2021-06-23  expires: 2023-06-23  usage: E
ssb  rsa3072/01265B6DAB6DEA96
     created: 2021-06-24  expires: never       usage: S
[ultimate] (1). Central Repo Test <central@example.com>
```

As you can see from above example, this key has 2 sub keys with ID `4656B4857C17C93B` and `01265B6DAB6DEA96`. The output also shows the creation time and expiration time. What's important here is the **usage** of these keys. `E` stands for Encryption so sub key `4656B4857C17C93B` is used for encryption only, `S` stands for Signing so sub key `01265B6DAB6DEA96` is used for Signing only.

If a primary key has a `S` sub key, it will use it for signing, otherwise itself will do signing job. So, in our example, we want to delete the sub key `01265B6DAB6DEA96`.

First select the sub key we want to delete, since its index is 2 (indices starts with 0), we run command:

```
gpg> key 2

sec  rsa3072/8190C4130ABA0F98
     created: 2021-06-23  expires: 2022-03-21  usage: SC
     trust: ultimate      validity: ultimate
ssb  rsa3072/4656B4857C17C93B
     created: 2021-06-23  expires: 2023-06-23  usage: E
ssb* rsa3072/01265B6DAB6DEA96
     created: 2021-06-24  expires: never       usage: S
[ultimate] (1). Central Repo Test <central@example.com>
```

As you can see from the output, the sub key `01265B6DAB6DEA96` is marked with `*`. Now delete it:

```
gpg> delkey
Do you really want to delete this key? (y/N) y

sec  rsa3072/8190C4130ABA0F98
     created: 2021-06-23  expires: 2022-03-21  usage: SC
     trust: ultimate      validity: ultimate
ssb  rsa3072/4656B4857C17C93B
     created: 2021-06-23  expires: 2023-06-23  usage: E
[ultimate] (1). Central Repo Test <central@example.com>
```

Tip

If you've already distributed your public key, it's better to revoke the sub signing key instead of deleting it, although either way you can make your primary key as the signing key. To revoke a sub key, use the `revkey` command instead of `delkey`.

Now that `01265B6DAB6DEA96` is not listed any more, the final step is saving our change:

```
gpg> save
```

That's it! Now you can test the change by signing a file, and then verify it. The output should contain something like:

```
gpg: Signature made *************************
gpg:                using RSA key [YOUR-PRIMARY-KEY-ID]
```