(PGP) Sign 3rd party artifacts in Eclipse Platform/IDE projects

Over the last months, many improvements have taken place in the Eclipse plugin releng world to simplify consumption and updates of dependencies. The 2 corner stones are the capability to include Maven artifacts directly in a Target Platform, and the support for PGP signatures as an alternative to jarsigner in p2 and Eclipse Simultaneous Release (so external artifacts don't need to be modified anymore).

On the development side, m2e adds to PDE's Target Platform editor the capability to easily include Maven artifacts, so they can easily be referenced later in MANIFEST.MF, feature.xml, category.xml and .product file.
On the build side, Tycho also reads this .target file and makes those Maven artifacts available as regular p2 installable units while building bundles/features/repository/products; it also provides a mojo that is capable of adding PGP signatures to artifacts in p2 repository.
On the consumer side, Eclipse p2 will read and verify PGP signatures when available while installing content, and let users decide whether to trust the installation or not.
On the organizational side, Simultaneous Release has allowed projects to contribute external artifacts that have a PGP signature produced by the contributing Eclipse.org project (as an alternative to jarsigner).

So no more repackaging/rebuild/resigning of external OSGi bundles, let's just take them from Maven without any modification and add PGP signatures to notify consumers that these artifacts are trusted by your Eclipse project!

While most parts are just implemented and work out of the box so you do not really need to care about it as an Eclipse plugin developer, how to produce PGP signatures is something that requires you to enable it; so let's talk a bit more about it:
Let's start with this example:
https://github.com/eclipse/tm4e/pull/379/commits/aba1fb103a341507d2dbbec81c16008216d5664a . As you can see, the overall idea is simply to run the tycho-gpg-plugin against a local p2 repository, as part of your Maven build, configuring it with the PGP key name. This mojo will run the underlying gpg command of the system to compute a PGP signature for each relevant artifact in the p2 repo, and will add that information into artifact metadata in the provided p2 repository. All this happens by a few more lines in pom.xml for your eclipse-repository:
```
<plugins>
                                        <plugin>
                                        <groupId>org.eclipse.tycho</groupId>
                                        <artifactId>tycho-gpg-plugin</artifactId>
                                        <version>${tycho-version}</version>
                                        <executions>
                                                <execution>
                                                        <id>pgpsigner</id>
```

```
                                                    <goals>
                                                        <goal>sign-p2-artifacts</goal>
                                                    </goals>
                                                    <configuration>

<keyname>27617A05E2DB057F</keyname> <!-- Replace it with your own key –>
                                                    </configuration>
                                                    </execution>
                                                </executions>
                                            </plugin>
```

That's the easy part. However for best results, it's recommended to use Tycho 3.0.0-SNAPSHOT.

The slightly trickier one is that you'll want to use the official project (private) key to sign the artifacts. That key is usually configured in the build environment; how to use it does differ from an environment to the other. Let's look at how to enable things in the Eclipse build infrastructure, more specifically to how to enable PGP signing at Eclipse.org from a Jenkinsfile.

A preliminary step is to request the Eclipse infrastructure team to set up the PGP key for your project and Jenkins instance. Here is an example of how to make such a request: https://gitlab.eclipse.org/eclipsefdn/helpdesk/-/issues/1191 .
Then, as documented in https://wiki.eclipse.org/Jenkins#Common_issues_with_GPG , the Jenkinsfile needs to include a few lines to load the PGP keystore. We won't repeat documentation here, but here is a link to how it looks like in TM4E Jenkinsfile: https://github.com/eclipse/tm4e/blob/master/Jenkinsfile#L14..L19 .
The next necessary thing is that signing with GPG requires you to give a passphrase to ensure the current user is trusted enough to use the private key. As tycho-pgp-plugin reads the `gpg.passphrase` Maven property, you "just" need to set this property to the right passphrase. To acquire the passphrase, you'll have to "wrap" your Maven execution into a Jenkins withCredentials block of the form `withCredentials([string(credentialsId: 'gpg-passphrase', variable: 'KEYRING_PASSPHRASE')]) { … }` that will set the $KEYRING_PASSPHRASE variable to the passphrase (this passphrase will be hidden in logs and so on, so unless you intentionally misuse it, there is almost no risk that this passphrase leaks). Then you can just pass the passphrase to the Maven execution with `-Dgpg.passphrase="$KEYRING_PASSPHRASE"`.

Then you'll have Jenkins properly loading the PGP key and passphrase, making them accessible to your Maven build so the tycho-gpg-plugin will use them to create PGP signatures for your artifacts (most specifically the ones that aren't jarsigned already, by default) and will be allowed to contribute those into SimRel.