Docs

[User Manual](http://docs.google.com/userguide/userguide.html)

[Guides and Tutorials](https://guides.gradle.org)

[DSL Reference](http://docs.google.com/dsl/)

[Javadoc](http://docs.google.com/javadoc/)

[Release Notes](http://docs.google.com/release-notes.html)

[Forums](https://discuss.gradle.org/)

[Training](https://gradle.org/training/)

[Try Gradle Enterprise](https://gradle.com/enterprise)

[PDF](http://docs.google.com/userguide/userguide.pdf)

* [User Manual Home](http://docs.google.com/userguide/userguide.html)
* [Release Notes](http://docs.google.com/release-notes.html)
* [Installing Gradle](http://docs.google.com/userguide/installation.html)
* [Tutorials](https://guides.gradle.org/)

### Reference

* [Groovy DSL Reference](http://docs.google.com/dsl/)
* [Gradle API Javadoc](http://docs.google.com/javadoc/)
* [Core Plugins](http://docs.google.com/userguide/plugin_reference.html)
* [Gradle & Third-party Tools](http://docs.google.com/userguide/third_party_integration.html)

### Getting Started

* [Creating New Gradle Builds](https://guides.gradle.org/creating-new-gradle-builds/)
* [Creating Build Scans](https://guides.gradle.org/creating-build-scans/)
* [Migrating From Maven](https://guides.gradle.org/migrating-from-maven/)

### Running Gradle Builds

* [Command-Line Interface](http://docs.google.com/userguide/command_line_interface.html)
* [Customizing Execution](#gjdgxs)
  + [Configuring the Build Environment](http://docs.google.com/userguide/build_environment.html)
  + [Configuring the Gradle Daemon](http://docs.google.com/userguide/gradle_daemon.html)
  + [Initialization Scripts](http://docs.google.com/userguide/init_scripts.html)
* [Directory Layout](http://docs.google.com/userguide/directory_layout.html)
* [Executing Multi-Project Builds](http://docs.google.com/userguide/intro_multi_project_builds.html)
* [Gradle Wrapper](http://docs.google.com/userguide/gradle_wrapper.html)
* [Troubleshooting](http://docs.google.com/userguide/troubleshooting.html)
* [Using Build Scans](https://docs.gradle.com/build-scan-plugin)
* [Enabling and Configuring the Build Cache](http://docs.google.com/userguide/build_cache.html)
* [Integrating Separate Gradle Builds (Composite Builds)](http://docs.google.com/userguide/composite_builds.html)

### Authoring Gradle Builds

* [Fundamentals](#30j0zll)
  + [Introducing the Basics of Build Scripts](http://docs.google.com/userguide/tutorial_using_tasks.html)
  + [Working with Tasks](http://docs.google.com/userguide/more_about_tasks.html)
  + [Learning More About Build Scripts](http://docs.google.com/userguide/writing_build_scripts.html)
  + [Working with Files](http://docs.google.com/userguide/working_with_files.html)
  + [Creating Custom Task Types](http://docs.google.com/userguide/custom_tasks.html)
  + [Using Gradle Plugins](http://docs.google.com/userguide/plugins.html)
  + [The Standard Gradle Plugins](http://docs.google.com/userguide/standard_plugins.html)
  + [Understanding the Build Lifecycle](http://docs.google.com/userguide/build_lifecycle.html)
  + [Working with Logging](http://docs.google.com/userguide/logging.html)
  + [Configuring Multi-Project Builds](http://docs.google.com/userguide/multi_project_builds.html)
* [Best Practices](#1fob9te)
  + [Authoring Maintainable Build Scripts](http://docs.google.com/userguide/authoring_maintainable_build_scripts.html)
  + [Organizing Gradle Projects](http://docs.google.com/userguide/organizing_gradle_projects.html)
  + [Optimizing Build Performance](https://guides.gradle.org/performance/)
  + [Using the Build Cache](https://guides.gradle.org/using-build-cache/)
* [Dependency Management](#3znysh7)
  + [Introduction to Dependency Management](http://docs.google.com/userguide/introduction_dependency_management.html)
  + [Dependency Management Terminology](http://docs.google.com/userguide/dependency_management_terminology.html)
  + [Dependency Types](http://docs.google.com/userguide/dependency_types.html)
  + [Repository Types](http://docs.google.com/userguide/repository_types.html)
  + [Declaring Dependencies](http://docs.google.com/userguide/declaring_dependencies.html)
  + [Declaring Repositories](http://docs.google.com/userguide/declaring_repositories.html)
  + [Inspecting Dependencies](http://docs.google.com/userguide/inspecting_dependencies.html)
  + [Managing Dependency Configurations](http://docs.google.com/userguide/managing_dependency_configurations.html)
  + [Managing Transitive Dependencies](http://docs.google.com/userguide/managing_transitive_dependencies.html)
  + [Dependency Locking](http://docs.google.com/userguide/dependency_locking.html)
  + [Troubleshooting Dependency Resolution](http://docs.google.com/userguide/troubleshooting_dependency_resolution.html)
  + [Customizing Dependency Resolution Behavior](http://docs.google.com/userguide/customizing_dependency_resolution_behavior.html)
  + [Dependency Cache Internals](http://docs.google.com/userguide/dependency_cache.html)
  + [Working with Dependencies](http://docs.google.com/userguide/working_with_dependencies.html)
* [Publishing Artifacts](http://docs.google.com/userguide/artifact_management.html)
* [C++ Projects](#2et92p0)
  + [Building Native Software](http://docs.google.com/userguide/native_software.html)
  + [Software Model Concepts](http://docs.google.com/userguide/software_model_concepts.html)
  + [Rule-based Model Configuration](http://docs.google.com/userguide/software_model.html)
  + [Implementing Model Rules in a Plugin](http://docs.google.com/userguide/rule_source.html)
  + [Extending the Software Model](http://docs.google.com/userguide/software_model_extend.html)
* [Java Projects](#tyjcwt)
  + [Building Java & JVM projects](http://docs.google.com/userguide/building_java_projects.html)
  + [Testing Java & JVM projects](http://docs.google.com/userguide/java_testing.html)
* [Advanced Techniques](#3dy6vkm)
  + [Configuring Tasks Lazily](http://docs.google.com/userguide/lazy_configuration.html)
  + [Developing Parallel Tasks](https://guides.gradle.org/using-the-worker-api/)
  + [Testing Your Build with TestKit](http://docs.google.com/userguide/test_kit.html)
  + [Using Ant from Gradle](http://docs.google.com/userguide/ant.html)
* [Sample Gradle builds](#1t3h5sf)
  + [Groovy DSL Samples](https://github.com/gradle/gradle/tree/master/subprojects/docs/src/samples)
  + [Kotlin DSL Samples](https://github.com/gradle/kotlin-dsl/tree/master/samples)

### Extending Gradle

* [Writing Custom Plugins](http://docs.google.com/userguide/custom_plugins.html)
* [Plugin Development Guides](https://gradle.org/guides/?q=Plugin+Development)

[Edit this page](https://github.com/gradle/gradle/edit/master/subprojects/docs/src/docs/userguide/)

# The Signing Plugin

Contents

[Usage](#4d34og8)

[Signatory credentials](#2s8eyo1)

[Using gpg-agent](#17dp8vu)

[Specifying what to sign](#3rdcrjn)

[Publishing the signatures](#26in1rg)

[Signing POM files](#lnxbz9)

The Signing Plugin adds the ability to digitally sign built files and artifacts. These digital signatures can then be used to prove who built the artifact the signature is attached to as well as other information such as when the signature was generated.

The Signing Plugin currently only provides support for generating [OpenPGP signatures](https://en.wikipedia.org/wiki/Pretty_Good_Privacy#OpenPGP) (which is the signature format [required for publication to the Maven Central Repository](http://central.sonatype.org/pages/requirements.html#sign-files-with-gpgpgp)).

[Usage](#4d34og8)

To use the Signing Plugin, include the following in your build script:

[Example: Using the Signing Plugin](#35nkun2)

**build.gradle**

plugins {  
 id 'signing'  
}

[Signatory credentials](#2s8eyo1)

In order to create OpenPGP signatures, you will need a key pair (instructions on creating a key pair using the [GnuPG tools](https://www.gnupg.org/) can be found in the [GnuPG HOWTOs](https://www.gnupg.org/documentation/howtos.html)). You need to provide the Signing Plugin with your key information, which means three things:

* The public key ID (The last 8 symbols of the keyId. You can use gpg -K to get it).
* The absolute path to the secret key ring file containing your private key. (Since gpg 2.1, you need to export the keys with command gpg --keyring secring.gpg --export-secret-keys > ~/.gnupg/secring.gpg).
* The passphrase used to protect your private key.

These items must be supplied as the values of the signing.keyId, signing.secretKeyRingFile, and signing.password properties, respectively.

| **✨** | Given the personal and private nature of these values, a good practice is to store them in the gradle.properties file in the user’s Gradle home directory (described in [System properties](http://docs.google.com/build_environment.html#sec:gradle_system_properties)) instead of in the project directory itself. |
| --- | --- |

signing.keyId=24875D73  
signing.password=secret  
signing.secretKeyRingFile=/Users/me/.gnupg/secring.gpg

If specifying this information (especially signing.password) in the user gradle.properties file is not feasible for your environment, you can source the information however you need to and set the project properties manually.

import org.gradle.plugins.signing.Sign  
  
gradle.taskGraph.whenReady { taskGraph ->  
 if (taskGraph.allTasks.any { it instanceof Sign }) {  
 // Use Java 6's console to read from the console (no good for  
 // a CI environment)  
 Console console = System.console()  
 console.printf "\n\nWe have to sign some things in this build." +  
 "\n\nPlease enter your signing details.\n\n"  
  
 def id = console.readLine("PGP Key Id: ")  
 def file = console.readLine("PGP Secret Key Ring File (absolute path): ")  
 def password = console.readPassword("PGP Private Key Password: ")  
  
 allprojects { ext."signing.keyId" = id }  
 allprojects { ext."signing.secretKeyRingFile" = file }  
 allprojects { ext."signing.password" = password }  
  
 console.printf "\nThanks.\n\n"  
 }  
}

Note that the presence of a null value for any these three properties will cause an exception.

[Using OpenPGP subkeys](#1ksv4uv)

OpenPGP supports subkeys, which are like the normal keys, except they’re bound to a master key pair. One feature of OpenPGP subkeys is that they can be revoked independently of the master keys which makes key management easier. A practical case study of how subkeys can be leveraged in software development can be read on the [Debian wiki](https://wiki.debian.org/Subkeys).

The Signing Plugin supports OpenPGP subkeys out of the box. Just specify a subkey ID as the value in the signing.keyId property.

[Using gpg-agent](#17dp8vu)

By default the Signing Plugin uses a Java-based implementation of PGP for signing. This implementation cannot use the gpg-agent program for managing private keys, though. If you want to use the gpg-agent, you can change the signatory implementation used by the Signing Plugin:

[Example: Sign with GnuPG](#44sinio)

**build.gradle**

signing {  
 useGpgCmd()  
 sign configurations.archives  
}

This tells the Signing Plugin to use the GnupgSignatory instead of the default [PgpSignatory](http://docs.google.com/javadoc/org/gradle/plugins/signing/signatory/pgp/PgpSignatory.html). The GnupgSignatory relies on the gpg2 program to sign the artifacts. Of course, this requires that GnuPG is installed.

Without any further configuration the gpg2 (on Windows: gpg2.exe) executable found on the PATH will be used. The password is supplied by the gpg-agent and the default key is used for signing.

[Gnupg signatory configuration](#2jxsxqh)

The GnupgSignatory supports a number of configuration options for controlling how gpg is invoked. These are typically set in gradle.properties:

[Example: Configure the GnupgSignatory](#z337ya)

**gradle.properties**

signing.gnupg.executable=gpg  
signing.gnupg.useLegacyGpg=true  
signing.gnupg.homeDir=gnupg-home  
signing.gnupg.optionsFile=gnupg-home/gpg.conf  
signing.gnupg.keyName=24875D73  
signing.gnupg.passphrase=gradle

signing.gnupg.executable

The gpg executable that is invoked for signing. The default value of this property depends on useLegacyGpg. If that is true then the default value of executable is "gpg" otherwise it is "gpg2".

signing.gnupg.useLegacyGpg

Must be true if GnuPG version 1 is used and false otherwise. The default value of the property is false.

signing.gnupg.homeDir

Sets the home directory for GnuPG. If not given the default home directory of GnuPG is used.

signing.gnupg.optionsFile

Sets a custom options file for GnuPG. If not given GnuPG’s default configuration file is used.

signing.gnupg.keyName

The id of the key that should be used for signing. If not given then the default key configured in GnuPG will be used.

signing.gnupg.passphrase

The passphrase for unlocking the secret key. If not given then the gpg-agent program is used for getting the passphrase.

All configuration properties are optional.

[Specifying what to sign](#3rdcrjn)

As well as configuring how things are to be signed (i.e. the signatory configuration), you must also specify what is to be signed. The Signing Plugin provides a DSL that allows you to specify the tasks and/or configurations that should be signed.

[Signing Publications](#3j2qqm3)

When publishing artifacts, you often want to sign them so the consumer of your artifacts can verify their signature. For example, the [Java plugin](http://docs.google.com/java_plugin.html#java_plugin) defines a component that you can use to define a publication to a Maven (or Ivy) repository using the [Maven Publish Plugin](http://docs.google.com/publishing_maven.html#publishing_maven) (or the [Ivy Publish Plugin](http://docs.google.com/publishing_ivy.html#publishing_ivy), respectively). Using the Signing DSL, you can specify that all of the artifacts of this publication should be signed.

[Example: Signing a publication](#1y810tw)

**build.gradle**

signing {  
 sign publishing.publications.mavenJava  
}

This will create a task (of type [Sign](http://docs.google.com/dsl/org.gradle.plugins.signing.Sign.html)) in your project named signMavenJavaPublication that will build all artifacts that are part of the publication (if needed) and then generate signatures for them. The signature files will be placed alongside the artifacts being signed.

[Example: Signing a publication output](#4i7ojhp)

**Output of** gradle signMavenJavaPublication

> gradle signMavenJavaPublication  
> Task :generatePomFileForMavenJavaPublication  
> Task :compileJava  
> Task :processResources  
> Task :classes  
> Task :jar  
> Task :javadoc  
> Task :javadocJar  
> Task :sourcesJar  
> Task :signMavenJavaPublication  
  
BUILD SUCCESSFUL in 0s  
8 actionable tasks: 8 executed

In addition, the above DSL allows to sign multiple comma-separated publications. Alternatively, you may specify publishing.publications to sign all publications, or use publishing.publications.matching { … } to sign all publications that match the specified predicate.

[Signing Configurations](#2xcytpi)

It is common to want to sign the artifacts of a configuration. For example, the [Java plugin](http://docs.google.com/java_plugin.html#java_plugin) configures a jar to build and this jar artifact is added to the archives configuration. Using the Signing DSL, you can specify that all of the artifacts of this configuration should be signed.

[Example: Signing a configuration](#1ci93xb)

**build.gradle**

signing {  
 sign configurations.archives  
}

This will create a task (of type [Sign](http://docs.google.com/dsl/org.gradle.plugins.signing.Sign.html)) in your project named signArchives, that will build any archives artifacts (if needed) and then generate signatures for them. The signature files will be placed alongside the artifacts being signed.

[Example: Signing a configuration output](#3whwml4)

**Output of** gradle signArchives

> gradle signArchives  
> Task :compileJava  
> Task :processResources  
> Task :classes  
> Task :jar  
> Task :signArchives  
  
BUILD SUCCESSFUL in 0s  
4 actionable tasks: 4 executed

[Signing Tasks](#2bn6wsx)

In some cases the artifact that you need to sign may not be part of a configuration. In this case you can directly sign the task that produces the artifact to sign.

[Example: Signing a task](#qsh70q)

**build.gradle**

task stuffZip (type: Zip) {  
 baseName = "stuff"  
 from "src/stuff"  
}  
  
signing {  
 sign stuffZip  
}

This will create a task (of type [Sign](http://docs.google.com/dsl/org.gradle.plugins.signing.Sign.html)) in your project named signStuffZip, that will build the input task’s archive (if needed) and then sign it. The signature file will be placed alongside the artifact being signed.

[Example: Signing a task output](#3as4poj)

**Output of** gradle signStuffZip

> gradle signStuffZip  
> Task :stuffZip  
> Task :signStuffZip  
  
BUILD SUCCESSFUL in 0s  
2 actionable tasks: 2 executed

For a task to be *signable*, it must produce an archive of some type, i.e. it must extend [AbstractArchiveTask](http://docs.google.com/dsl/org.gradle.api.tasks.bundling.AbstractArchiveTask.html). Tasks that do this are the [Tar](http://docs.google.com/dsl/org.gradle.api.tasks.bundling.Tar.html), [Zip](http://docs.google.com/dsl/org.gradle.api.tasks.bundling.Zip.html), [Jar](http://docs.google.com/dsl/org.gradle.api.tasks.bundling.Jar.html), [War](http://docs.google.com/dsl/org.gradle.api.tasks.bundling.War.html) and [Ear](http://docs.google.com/dsl/org.gradle.plugins.ear.Ear.html) tasks.

[Conditional Signing](#1pxezwc)

A common usage pattern is to require the signing of build artifacts only under certain conditions. For example, you may not need to sign artifacts for non-release versions. To achieve this, you can specify the condition as an argument of the required() method.

[Example: Specifying when signing is required](#49x2ik5)

**build.gradle**

version = '1.0-SNAPSHOT'  
ext.isReleaseVersion = !version.endsWith("SNAPSHOT")  
  
signing {  
 required { isReleaseVersion && gradle.taskGraph.hasTask("uploadArchives") }  
 sign configurations.archives  
}

In this example, we only want to require signing if we are building a release version and we are going to publish it. Because we are inspecting the task graph to determine if we are going to be publishing, we must set the signing.required property to a closure to defer the evaluation. See [SigningExtension.setRequired(java.lang.Object)](http://docs.google.com/javadoc/org/gradle/plugins/signing/SigningExtension.html#setRequired-java.lang.Object-) for more information.

If the required condition does not hold true, artifacts will only be signed if signatory credentials are configured. Alternatively, you may want to skip signing entirely whether or not signatory credentials are available. If so, you can configure the [Sign](http://docs.google.com/dsl/org.gradle.plugins.signing.Sign.html) tasks to be skipped, for example by attaching a predicate using the onlyIf() method shown in the following example:

[Example: Specifying when signing is skipped](#2p2csry)

**build.gradle**

tasks.withType(Sign) {  
 onlyIf { isReleaseVersion }  
}

[Publishing the signatures](#26in1rg)

When signing [publications](#3j2qqm3), the resultant signature artifacts are automatically added to the corresponding publication. Thus, when publishing to a repository, e.g. by executing the publish task, your signatures will be distributed along with the other artifacts without any additional configuration.

When signing [configurations](#2xcytpi) and [tasks](#2bn6wsx), the resultant signature artifacts are automatically added to the signatures and archives dependency configurations. This means that if you want to upload your signatures to your distribution repository along with the artifacts you simply execute the uploadArchives task.

[Signing POM files](#lnxbz9)

| **✨** | This section covers signing POM files for the *original* publishing mechanism available in Gradle 1.0. The POM file generated by the *new* Maven publishing support provided by the [Maven Publishing plugin](http://docs.google.com/publishing_maven.html#publishing_maven) is automatically signed if the corresponding publication is [specified to be signed](#3j2qqm3). |
| --- | --- |

When deploying signatures for your artifacts to a Maven repository, you will also want to sign the published POM file. The Signing Plugin adds a signing.signPom() (see [SigningExtension.signPom(org.gradle.api.artifacts.maven.MavenDeployment, groovy.lang.Closure)](http://docs.google.com/dsl/org.gradle.plugins.signing.SigningExtension.html#org.gradle.plugins.signing.SigningExtension:signPom(org.gradle.api.artifacts.maven.MavenDeployment,%20groovy.lang.Closure))) method that can be used in the beforeDeployment() block in your upload task configuration.

[Example: Signing a POM for deployment](#147n2zr)

**build.gradle**

uploadArchives {  
 repositories {  
 mavenDeployer {  
 beforeDeployment { MavenDeployment deployment -> signing.signPom(deployment) }  
 }  
 }  
}

When signing is not required and the POM cannot be signed due to insufficient configuration (i.e. no credentials for signing) then the signPom() method will silently do nothing.

Docs

* [User Manual](http://docs.google.com/userguide/userguide.html)
* [DSL Reference](http://docs.google.com/dsl/)
* [Release Notes](http://docs.google.com/release-notes.html)
* [Javadoc](http://docs.google.com/javadoc/)

News

* [Blog](https://blog.gradle.org/)
* [Newsletter](https://newsletter.gradle.com/)
* [Twitter](https://twitter.com/gradle)

Products

* [Build Scans](https://gradle.com/build-scans)
* [Build Cache](https://gradle.com/build-cache)
* [Enterprise Docs](https://gradle.com/enterprise/resources)

Get Help

* [Forums](https://discuss.gradle.org/c/help-discuss)
* [GitHub](https://github.com/gradle/)
* [Training](https://gradle.org/training/)
* [Services](https://gradle.org/services/)

Subscribe for important Gradle updates and news

Subscribe

By entering your email, you agree to our [Terms](https://gradle.org/terms/) and [Privacy Policy](https://gradle.org/privacy/), including receipt of emails. You can unsubscribe at any time.

© [Gradle Inc.](https://gradle.com) 2018 All rights reserved.

[Careers](https://gradle.com/careers) | [Privacy](https://gradle.org/privacy) | [Terms of Service](https://gradle.org/terms) | [Contact](https://gradle.org/contact/)