Android studio apk signature verification failed

The correct adb installation verifies the app's signature. If the app was not installed on the device before the package manager can only verify that the app has been signed, the app signer cannot be verified by the package manager. Only if you install an update for an app that is already installed before the package manager can verify whether the new app signature matches the app that is already installed. In case the update.apk to be installed has a different signature, you will receive an error message after you install adb so that the signatures do not match the previously installed version, I am not sure that this will also happen in case you installed the old app before using adb uninstall -k. I assume (but I've never tested it) that the signature is also checked and compared and that you can only install an apk with a corresponding signature. Otherwise you could access any private app data on your phone by replacing the app with a custom app. This would be a serious security vulnerability. If you also want to check the downloaded apk files you can use apksigner included with the Android SDK. The necessary steps are described in this answer that I wrote some time ago. In particular, publisher verification (check which other apps have been signed by the same entity) using androidobservatory.org and apkmirror.com interest you. forward-lock to my knowledge is if you want to add an app to the current version of the device firmware, then mark an app as potentially incompatible with future android versions. The apksigner tool, available in revision 24.0.3 and later of the Android SDK compilation tools, allows you to sign APKs and confirm that the signature of an APK will be verified correctly on all versions of the Android platform supported by those APKs. This page presents a short guide for using the tool and serves as a reference for the different command-line options supported by the tool. For a more complete description of using the apksigner tool to sign APKs, see the app's signature guide. Warning: If you sign the APK using apksigner and make further changes to the APK, the APK signature is invalidated. Therefore, you need to use tools such as zipalign before signing the APK. Syntax Sign an APK The syntax for signing an APK using the apksigner tool is as follows: apksigner sign --ks keystore.jks | --key key.pk8 --cert cert.x509.pem [signer_options] app-name.apk When signing an APK using the apksigner tool, you must provide the signer's private key and certificate. You can include these in two different ways: specify a KeyStore file using the --ks option. Specify the private key file and certificate file separately using the --key and --cert options, respectively. The private key file must use the PKCS #8 format, and the certificate file must use the X.509 format. Usual Usual sign an APK using only one signer. In case you need to sign an APK using multiple signers, use the --next-signer option to separate the set of general options to apply to each signer: apksigner sign [signer_1_options] --app name [signer_2_options].apk Verify the signing an APK The syntax for confirming that an APK signature will be verified correctly on supported platforms is as follows: apksigner verify [options] app-name.apk Rotate signature keys Syntax to rotate a signature certificate derivation. or a new sequence of signatures, is as follows: $ apksigner rotate --in /path/to/existing/lineage \ --out /path/to/new/file \ --old-signer --ks old-signer-jks \ --new-signer --ks new-signer-jks Options The following lists include the set of options for each command supported in the apksigner tool. General Options Signature command The following options specify the basic settings to apply to a signer: --out The path in &lt;apk-filename&gt;to which you want to save the signed APK. If this option is not explicitly provided, the APK package is signed in place, overwriting the input APK file. --min-sdk-version &lt;integer&gt;The lowest Android framework api level used by apksigner to confirm that the APK signature will be verified. Higher values allow the tool to use stronger security parameters when signing the app, but limit the availability of the APK to devices running newer versions of Android. By default, apksigner uses the value of the minSdkVersion attribute from the app manifest file. --max-sdk-version &lt;integer&gt;The highest level of Android framework APIs used by apksigner to confirm that the APK signature will be verified. By default, the tool uses the highest possible API level. --v1-signing-enabled &lt;true |= false=&gt;Determines whether apksigner signs the supplied APK package using the traditional JAR-based signature scheme. By default, the tool uses the values of --min-sdk-version and --max-sdk-version to decide when to apply this signature scheme. --v2-signing-enabled &lt;true |= false=&gt;Determines whether apksigner signs the supplied APK package using the V2 APK signature scheme. By default, the tool uses the values of --min-sdk-version and --max-sdk-version to decide when to apply this signature scheme. --v3-signing-enabled &lt;true |= false=&gt;Determines whether apksigner signs the supplied APK package using the V3 APK signature scheme. By default, the tool uses the values of --min-sdk-version and --max-sdk-version to decide when to apply this signature scheme. --v4-signing-enabled &lt;true |= |= only=&gt;Determines whether apksigner signs the supplied APK package using the APK v4 signature scheme. This schema produces a signature in a separate file (apk-name.apk.idsig). If true and the APK is not signed, a v2 or v3 signature is generated based on the values of --min-sdk-version and --max-sdk-version. The command then produces the file idsig&lt;/true&gt; &lt;/true&gt; &lt;/true&gt; &lt;/true&gt; &lt;/integer&gt; &lt;/integer&gt; &lt;/apk-filename&gt; &lt;/apk-filename&gt; on the content of the signed APK. Use only to generate only the v4 signature without changing the APK and signatures it had before the call; fails only if the APK does not already have a v2 or v3 signature or if the signature used a key other than the one provided for the current call. By default, the tool uses the values of --min-sdk-version and --max-sdk-version to decide when to apply this signature scheme. --v4-no-merkle-tree By default, the idsig file includes a full merkle tree of the APK file. With this flag, apksigner produces an APK Signature Scheme v4 .idsig file without the built-in full Merkle tree. This option reduces the size of the signature file, but forces any tool that needs that structure to recalculate it again or call the apksigner tool again. -v, --verbose Use verbose mode of verbose output. Signer options The following options specify the configuration of a particular signer. These options are not necessary if you sign the app using only one signer. --next signer &lt;signer-options&gt;Used to specify several general options for each signer. --v1-signer-name The base name for files that &lt;basename&gt;are the JAR-based signature for the current signer. By default, apksigner uses the KeyStore key alias or the base key file name for this signer. Key and certificate options The following options specify the signer's private key and certificate: --ks The private key and &lt;filename&gt;signer's certificate chain reside in the specified Java-based KeyStore file. If the file name is set to NONE, the KeyArchive containing the key and certificate does not require the addition of a file, as is the case with some keystores #11 PKCS. --ks-key-alias &lt;alias&gt;Name of the alias that represents the signer's private key and certificate data within the KeyStore. If keystore associated with the signer contains multiple keys, you must specify this option. --ks-pass &lt;input-format&gt;Password for keystore containing the signer's private key and certificate. You must provide a password to open a KeyStore. The apksigner tool supports the following formats: pass:&lt;password&gt; – Password that comes in line with the rest of the apksigner sign command. env: – The password is stored in the data environment variable&lt;name&gt; . file: – The password is&lt;filename&gt; stored as a single line in the specified file. stdin – The password is provided as a single line in the standard input stream. This is the default behavior for --ks-pass. Note: If you include multiple passwords in the same file, specify them on separate lines. Tool associa le password ai firmatari di un APK in base all'ordine in cui si specificano i firmatari. Se sono state fornite due password per un firmatario, apksigner interpreta la prima password come password KeyStore e la seconda come password chiave. --pass-encoding &lt;charset&gt;Include il carattere specificato&lt;/charset&gt; &lt;/filename&gt; &lt;/name&gt; &lt;/password&gt; &lt;/input-format&gt; &lt;/alias&gt; &lt;/filename&gt; &lt;/basename&gt; &lt;/signer-options&gt; &lt;/signer-options&gt; (for example, ibm437 or utf-8) when you try to manage passwords that contain non-ASCII characters. Keytool often encrypts keytors by converting the password using the console's default charset. By default, apksigner attempts to decrypt using different forms of the password: the Unicode module, the form encoded using the default JVM charset, and, on Java 8 and earlier, the form encoded using the console's default charset. In Java 9, apksigner cannot detect the console character set. Therefore, you may need to specify --pass-encoding when using a non-ASCII password. You may also need to specify this option with keytors created on a different operating system or in different locales. --key-pass &lt;input-format&gt;The password for the signer's private key, which is required if the private key is password protected. The apksigner tool supports the following formats: pass:&lt;password&gt; – Password that comes in line with the rest of the apksigner sign command. env: – The password is stored in the data environment variable&lt;name&gt; . file: – The password is&lt;filename&gt; stored as a single line in the specified file. stdin – The password is provided as a single line in the standard input stream. This is the default behavior for --key-pass. Note: If you include multiple passwords in the same file, specify them on separate lines. The apksigner tool associates passwords with signatories of an APK in the order in which you specify signatories. If you have provided two passwords for a signer, apksigner interprets the first password as the KeyStore password and the second password as the key password. --ks-type &lt;algorithm&gt;Type or algorithm associated with the KeyStore that contains the signer's private key and certificate. By default, apksigner uses the type defined as the keystore.type constant in the Security property file. --ks-provider-name &lt;name&gt;Name of the JCA provider to use when requesting the signer's keyer implementation. By default, apksigner uses the highest priority provider. --ks-provider-class &lt;class-name&gt;Full class name of the JCA provider to use when requesting the signer's KeyStore implementation. This option serves as an alternative for --ks-provider-name. By default, apksigner uses the provider specified with the --ks-provider-name option. --ks-provider-arg &lt;value&gt;String value to pass as an argument to the JCA Provider class constructor; the class itself is defined with the --ks-provider-class option. By default, apksigner uses the class's 0-argument constructor. --key of the file that contains the signer's private key. This file must use the PKCS format #8 DER. If the key is password protected, apksigner requires the password using standard input unless you specify a different type of input format using the --key-pass option. --cert &lt;filename&gt;Name of the file that contains the signer's certificate chain. This file must&lt;/filename&gt; &lt;/filename&gt; &lt;/value&gt; &lt;/class-name&gt; &lt;/name&gt; &lt;/algorithm&gt; &lt;/filename&gt; &lt;/name&gt; &lt;/password&gt; &lt;/input-format&gt; &lt;/input-format&gt; X.509 PEM or DER format. Verify command --print-certs Shows information about APK signing certificates. --min-sdk-version &lt;integer&gt;The lowest Android framework api level used by apksigner to confirm that the APK signature will be verified. Higher values allow the tool to use stronger security parameters when signing the app, but limit the availability of the APK to devices running newer versions of Android. By default, apksigner uses the value of the minSdkVersion attribute from the app manifest file. --max-sdk-version &lt;integer&gt;The highest level of Android framework APIs used by apksigner to confirm that the APK signature will be verified. By default, the tool uses the highest possible API level. -v, --verbose Use verbose mode of verbose output. -Werr Treat warnings as errors. Examples Sign an APK Sign an APK using release.jks, which is the only key in the KeyStore: sign $ apksigner --ks release.jks app.apk Sign an APK using a private key and certificate. stored as separate files: $ apksigner sign --key release.pk8 --cert release.x509.pem app.apk Sign an APK using two keys: $ apk signal sign --ks first-release-key.jks --next-signer --ks second key-version.jks app.apk Verify aPK signature Check if APK signatures should be confirmed valid on all Android platforms supported by the APK: $ apksigner verifies the app.apk Check if APK signatures should be confirmed as valid on Android 4.0.3 (API level 15) and higher versions: $ apksigner verification -min-sdk-version 15 app.apk Rotate signature keys Enable a signature certificate lineage that supports key rotation : $ apksigner rotate --out /path/to/new/file --old-signer \ --ks release.jks --new-signer --ks release2.jks Rotate signature keys again: $ apksigner rotateer --in /path/to/existing/lineage \ --out /path/to/new/file --old-signer --ks release2.jks \ --new-signer --ks release3.jks release3.jks&lt;/integer&gt; &lt;/integer&gt;