

(<https://www.misterpki.com/author/misterpki/>).

A pkcs12 keystore is commonly used for both S/MIME User Certificates and SSL/TLS Server Certificates (<https://www.misterpki.com/what-are-ssl-certificates/>). The keystore may contain both private keys and their corresponding certificates with or without a complete chain. The keystore's purpose is to store the credential of an identity, being a person, client, or server. Common file extensions include .p12 or .pfx for clarity, but may be anything you choose.

If you are in the market of purchasing a new SSL Certificate, start here (<https://www.thesslstore.com/?aid=52914810>).

RFC 7292 goes into much much much more detail about the PKCS #12 standard:
<https://tools.ietf.org/html/rfc7292>
(<https://tools.ietf.org/html/rfc7292>).

Unfortunately, there is not 100% coverage in all commands for maintaining PKCS #12 keystores in either OpenSSL or the Java Keytool so you must use both for comprehensive coverage of all the functions for maintaining your keystore.

The PKCS12 keystore is non-proprietary unlike the JKS and is becoming the most commonly used format. In fact, if you choose to generate a JKS keystore with the Java Keytool you will receive the following warning:

Warning:

The JKS keystore uses a proprietary format. It is recommended to migrate to PKCS12 which is an industry standard format

With that said, this post strives to provide examples to common commands used to create and manage PKCS12 keystores that will hopefully make your life on the job a bit easier.

How do I create a keystore with a self-signed certificate using the java keytool?

Using the Java Keytool, run the following command to create the keystore with a self-signed certificate:

```
keytool -genkey \  
  -alias somealias \  
  -keystore keystore.p12 \  
  -storetype PKCS12 \  
  -keyalg RSA \  
  -storepass somepass \  
  -validity 730 \  
  -keysize 4096
```

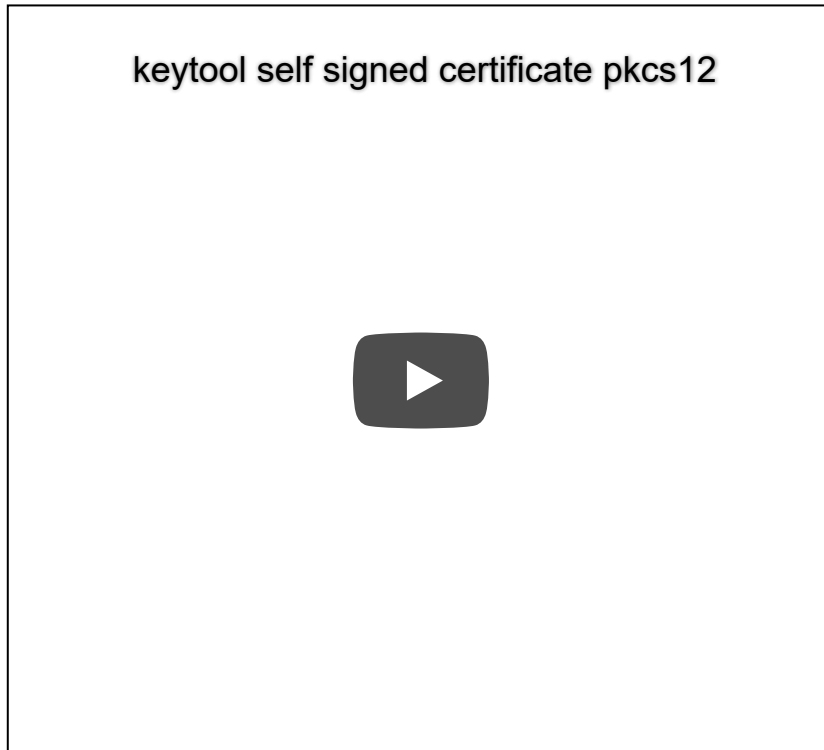
Keystore generation option breakdown:

Keytool option	Description
-genkey	Generate keystore
-alias	Alias of the generated private key entry
-keystore	Keystore file to be created
-storetype	Type of keystore. PKCS12 in this example
-keyalg	Key algorithm of key entry to be generated
-storepass	Password to set on both the key entry and keystore
-validity	Validity of the certificate associated with the key entry
-keysize	Size of the generated private key in bits

Keytool genkey options for PKCS12 keystore

Watch the video below for a visual example of generating a pkcs12 keystore with keytool. Or head

over to our focused post on how to [generate a keystore with the java keytool](https://www.misterpki.com/java-keytool-generate-keystore/) (<https://www.misterpki.com/java-keytool-generate-keystore/>).



java keytool generate keystore and self-signed certificate

How do I create a PKCS12 keystore from an existing private key and certificate using openssl?

While some configurations require the certificate and private key to be in separate files with pointers to them, it is becoming more common for configuration to point to a keystore instead. To create the keystore from an existing private key and certificate run the

from an existing private key and certificate, run the following command:

```
openssl pkcs12 \  
-export \  
-in certificate.pem \  
-inkey key.pem \  
-out keystore.p12
```

OpenSSL Option	Description
pkcs12	Create pkcs12 formatted keystore
-export	Export the keystore to a file
-in	The existing certificate file
-inkey	The existing key file
-out	The name of the newly created pkcs12 keystore

OpenSSL Option	Description
OpenSSL options for creating PKCS12 keystore from an existing private key and certificate	

How do I convert a JKS keystore to PKCS12?

This example uses the **importkeystore** command. For more details on this command head over to our focused post on [importing an existing keystore into another keystore](https://www.misterpki.com/import-keystore-into-another-keystore/) (<https://www.misterpki.com/import-keystore-into-another-keystore/>). To convert a Java Keystore to a PKCS #12 Keystore (.jks to .p12), run the following command:

```
keytool -importkeystore \  
-srckeystore keystore.jks \  
-destkeystore keystore.p12 \  
-srcstoretype JKS \  
-deststoretype PKCS12 \  
-deststorepass password \  
-srcalias alias \  
-destalias alias
```

Keytool Options	Description
-importkeystore	Import existing keystore into new keystore
-srckeystore	The keystore to be imported
-destkeystore	The keystore to accept the import
-srcstoretype	The type of keystore to be imported
-deststoretype	The type of keystore to accept the import
-deststorepass	The password of the new keystore
-srcalias	The alias to be imported
-destalias	The alias to import to

JKS to PKCS12 keystore conversion example

How do I convert a PKCS12 keystore to JKS?

To convert a PKCS12 keystore to JKS, run the following command:

```
keytool -importkeystore \  
-srckeystore example.p12 \  
-srcstoretype PKCS12 \  
-destkeystore example.jks \  
-deststoretype JKS
```

[Read more about using java keytool to import a keystore into another keystore.](https://www.misterpki.com/import-keystore-into-another-keystore/)
(<https://www.misterpki.com/import-keystore-into-another-keystore/>).

How do I change the password of a PKCS12 keystore?

To change the password of a PKCS #12 keystore (make sure to also change the password of the key, if not, the keystore will be corrupt), run the following commands:

First, change the keystore password:

```
keytool -storepasswd -keystore  
keystore.p12
```

Finally, change the key password in that keystore for each alias:


```
keytool -keypasswd -alias alias -  
keystore keystore.p12
```

How do I change an alias name in a keystore?

When generating a keystore, the default alias is 1 if not explicitly set. This default value may vary based on the software used to generate the keystore. We have a focused post on changing an alias [here](https://www.misterpki.com/keytool-alias/) (<https://www.misterpki.com/keytool-alias/>) for even more details on this command. To change the alias, run the following command:

```
keytool -changealias -keystore  
keystore.p12 -alias alias
```

How do I list the contents of a keystore?

It is useful and recommended to verify any changes made to a keystore. The simplest way is to just list the contents. To list the contents of the PKCS #12 keystore run the following command:

```
keytool -list -v -keystore  
keystore.p12
```

How do I extract a private key from a keystore using openssl?

Some software requires a stand alone private key instead of a keystore for authentication, signing, etc. To extract the private key from a keystore, run the following command:

```
openssl pkcs12 -in keystore.p12 -  
nocerts -nodes
```

Note that secret keys are not supported with openssl in a pkcs12 keystore. If you attempt to extract a secret key entry you will receive the following exception: **Warning unsupported bag type: secretBag.**

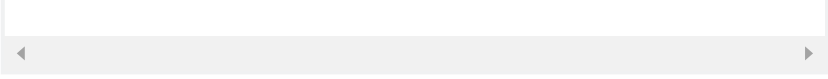
View the video on extracting a private key from a keystore with OpenSSL for a step by step walk through.

Extract a private key from a pkcs12 keystore with openssl

How do I extract certificates from a keystore using openssl?

Similar to requiring a stand alone key, some software requires stand alone certificate files to be used instead of a keystore. To extract a certificate or certificate chain from a PKCS12 keystore using openssl, run the following command:

```
openssl pkcs12 -in example.p12 -nokeys
```



Where `-in example.p12` is the keystore and `-nokeys` means only extract the certificates and not the keys.

How do I update the trust chain in an existing keystore for a specific keystore entry?

You may find it useful or necessary to update a trust chain to an existing keystore entry. For example, in the event of an expiring trust chain due to a cross signed root or intermediate, you may have an expiring chain installed and need to replace it (like with the [AddTrust root expiration](https://sectigo.com/resource-library/may-30-addtrust-root-expiration-explained) (<https://sectigo.com/resource-library/may-30-addtrust-root-expiration-explained>)) with [Sectigo](https://www.sectigostore.com?aid=52914810) (<https://www.sectigostore.com?aid=52914810>). To update the trust chain for a given alias in a pkcs12 keystore, run the following command:

```
keytool -import \  
-trustcacerts \  
-alias alias to be updated \  

```

```
-file chain.pem \  
-keystore keystore.p12
```

Where `-trustcacerts` means the trust chain is being added to the existing entry, `-alias alias_to_be_updated` is the entry being updated, `-file chain.pem` is the complete certificate chain including the root, and `-keystore keystore.p12` is the keystore being updated.

If you encounter Error: “java.lang.exception: failed to establish chain from reply” it is likely you have not included the correct chain or the complete chain, including the root. Also, check the formatting of the chain as it is easy to miss a character in the header and/or footer of each certificate in the chain.

Keystore Exceptions – PKCS12, JKS, or any type

When operating on a keystore, you will likely enter invalid input or find your keystore in a corrupt state at some point. Here are some common exceptions you may see.

When listing a keystore (and likely other operations), you may encounter this error:

keytool error: java.security.KeyStoreException: This keystore does not support probing and must be loaded with a specified type

```
keytool error:  
java.security.KeyStoreException:  
This keystore does not support  
probing and must be loaded with a  
specified type  
java.security.KeyStoreException:  
This keystore does not support  
probing and must be loaded with a  
specified type  
at  
java.base/java.security.KeyStore.g  
etInstance(KeyStore.java:1816)  
at  
java.base/java.security.KeyStore.g  
etInstance(KeyStore.java:1687)  
at  
java.base/sun.security.tools.keyto  
ol.Main.doCommands(Main.java:924)  
at  
java.base/sun.security.tools.keyto  
ol.Main.run(Main.java:409)  
at  
java.base/sun.security.tools.keyto  
ol.Main.main(Main.java:402)
```

To move past this error, simply specify the keystore type with **-storetype PKCS12** (or the store type of your keystore) in your command.

Create PKCS12 keystore with Java

There are many reasons you may need to generate a

keystore with Java instead of on the command line. As part of your Java application you may be issuing certificates, keys, keystores, etc. and need to generate a keystore programmatically. Additionally it may be useful to generate a new keystore in a test environment to not have a static keystore sitting around that will likely contain expired certificates at some point.

Here is some Java code to programmatically create the Keystore. For the complete example, review the GitHub project at <https://github.com/misterpki/generate-keystore> (<https://github.com/misterpki/generate-keystore>).

```
public static KeyStore  
generatePKCS12KeyStore(final  
String password)
```

```

        throws KeyStoreException,
        NoSuchAlgorithmException,
        IOException, CertificateException,
        OperatorCreationException
    {
        final KeyStore keyStore =
        KeyStore.getInstance("PKCS12");
        keyStore.load(null,
        password.toCharArray());

        // Create Symmetric key entry
        final KeyGenerator
        aesGenerator =
        KeyGenerator.getInstance("AES");
        aesGenerator.init(128);
        final KeyStore.SecretKeyEntry
        aesSecretKey = new
        KeyStore.SecretKeyEntry(aesGenerat
        or.generateKey());
        final
        KeyStore.ProtectionParameter
        aesSecretKeyPassword =
            new
        KeyStore.PasswordProtection(passwo
        rd.toCharArray());
        // Add symmetric key to
        keystore
        keyStore.setEntry("symm-key",
        aesSecretKey,
        aesSecretKeyPassword);

        // Create Asymmetric key pair
        final KeyPair asymmetricKeys =
        KeyPairGenerator.getInstance("RSA"
        ).generateKeyPair();
        final KeyStore.PrivateKeyEntry
        privateKey =
            new
        KeyStore.PrivateKeyEntry(
        asymmetricKeys.getPrivate())
    }

```



```
asymmetricKeys.getPrivate(),
        new X509Certificate[]
{generateX509Certificate(asymmetricKeys)});
    final
    KeyStore.ProtectionParameter
    privateKeyPassword =
        new
    KeyStore.PasswordProtection(password.toCharArray());
    // Add asymmetric key to
    keystore
    keyStore.setEntry("asymm-key",
    privateKey, privateKeyPassword);

    return keyStore;
}
```

Another example (not recommended) but for general demonstration.

```
package com.misterpki;

import java.io.FileOutputStream;
```

```
import java.io.IOException;
import
java.security.InvalidKeyException;
import java.security.KeyStore;
import
java.security.KeyStoreException;
import
java.security.NoSuchAlgorithmException;
import
java.security.NoSuchProviderException;
import
java.security.SignatureException;
import
java.security.cert.CertificateException;
import
java.security.cert.X509Certificate;
import
sun.security.tools.keytool.CertAndKeyGen;
import sun.security.x509.X500Name;

public class Main {

    public static void main(String[]
args) throws Exception {
        try (final FileOutputStream
stream = new
FileOutputStream("keyStore.p12"))
        {

            generatePKCS12().store(stream,
"changeit".toCharArray());

        }

        private static KeyStore
generatePKCS12() throws
```

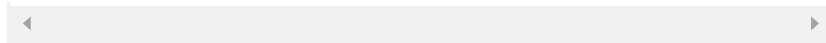
```
generatePKCS12() throws
NoSuchProviderException,
    NoSuchAlgorithmException,
    KeyStoreException,
    InvalidKeyException,
    IOException,
    CertificateException,
    SignatureException
{
    final KeyStore keyStore =
KeyStore.getInstance("PKCS12");
    keyStore.load(null, new
char[0]);
    final CertAndKeyGen
certGenerator = new
CertAndKeyGen("RSA",
"SHA256WithRSA", null);
    certGenerator.generate(4096);

    //validity of 1 year
    final long validSecs = (long)
365 * 24 * 60 * 60;
    final X509Certificate
certificate =
certGenerator.getSelfCertificate(
        new
X500Name("CN=some_name,O=some_org,
L=some_city,C=some_country"),
validSecs);

    keyStore.setKeyEntry("alias",

certGenerator.getPrivateKey(),
        "changeit".toCharArray(),
        new X509Certificate[]
{certificate});



    return keyStore;
}
}
```



Conclusion

Please comment with questions and suggestions for additional pkcs12 openssl and java keytool commands that may be helpful so that we may provide as valuable content as possible.

Read all blog content.
(<https://www.misterpki.com/blog/>).

 
(h (h
tt tt
ps ps
:/ :/
/ /
w w
w w
w. w.
t y
w o
itt ut
er u
.c b
o e.
m c
/ o
m m
is /c
te h
rp a
ki n
) n
el
/
U
C
h
Z
7
o
31
2
m
6
m
e
nl
T
6
O
A
h
n
p
A
A)

Pages

[About Mister PKI \(https://www.misterpki.com/about-mister-pki/\)](https://www.misterpki.com/about-mister-pki/)

[Blog \(https://www.misterpki.com/blog/\)](https://www.misterpki.com/blog/)

[Compare and Buy Affordable PKI Certificates \(https://www.misterpki.com/\)](https://www.misterpki.com/)

[Contact Us \(https://www.misterpki.com/contact/\)](https://www.misterpki.com/contact/)

[Full Disclosure \(https://www.misterpki.com/full-disclosure/\)](https://www.misterpki.com/full-disclosure/)

[Privacy Policy \(https://www.misterpki.com/privacy-policy/\)](https://www.misterpki.com/privacy-policy/)

[SSL Tools – Certificate Decoder and Certificate Checker \(https://www.misterpki.com/ssl-tools/\)](https://www.misterpki.com/ssl-tools/)

Copyright © 2021