Oracle® Jipher

User's Guide

Release 10
G18237-03
January 2015

Oracle Jipher User's Guide, Release 10

G18237-03

Contents

Preface

Audience v
Documentation Accessibility v
Diversity and Inclusion v

1 What is Oracle Jipher?

The Java Cryptographic Architecture (JCA), Engine Classes, and Providers 1-2
FIPS 140 1-3
Jipher Artifacts 1-3
Independence from and Coexistence with Other Instances of OpenSSL 1-3

2 Downloading and Deploying Jipher

Downloading Jipher 2-1
Jipher Deployment 2-1
Check the Version of Your Jipher Deployment 2-1

3 Configuring an Application to Achieve FIPS 140 Compliance with Jipher

Provider Registration 3-2
    Registering Jipher as the Most Preferred Provider 3-2
    Registering the SUN and SunRsaSign Providers for JAR File Signature Verification 3-4
    Registering Other Security Providers 3-5
        Delayed Provider Selection 3-7
Configure the SunJSSE Provider to Select Jipher to Provide Cryptography 3-7
    Permit Access to Internal JDK API Classes 3-8
    Limit Protocol Versions, Cipher Suites, Key Material, and Named Groups 3-8
        Ensure That Clients and Servers Only Use TLS 1.2 and TLS 1.3 3-8
        Ensure TLS Cipher Suites Use FIPS 140 Algorithms from Jipher 3-9
        Configure SSLContext with Key Material That Provides at Least 112-Bits of Security 3-10
        Configure the Size of Ephemeral Diffie-Hellman Keys 3-10
        Ensure the JSSE Only Uses FIPS 140 Approved Named Groups 3-10
Explicitly Selecting Jipher to Provide Cryptography 3-10

Implicitly Selecting Jipher to Provide Cryptography 3-11
Configuring Jipher Through System Properties 3-11
FIPS 140 Enforcement Policy 3-14

4 Applying NIST Guidelines for TLS

Configure Minimum Key Sizes for Certificate Path Validation 4-1
Enable Revocation Checking 4-1

5 Jipher Performance Optimizations

KeyManager Selection 5-1
Elliptic Curve Key Pair Generation 5-1

6 Secure Coding Guidance

Avoiding Unnecessary In-Memory Buffering of Plaintext 6-1

7 Jipher Diagnostics

Confirming That a Java Application Is Using Jipher 7-1
Keeping Track of Security Provider Usage with the jdk.SecurityProviderService Java Flight Recorder (JFR) Event 7-2
Reporting the Enforcement of FIPS 140 Restrictions 7-2
Reporting Misconfiguration 7-3
Reporting Abnormal Operation in OpenSSL Native Code 7-3

8 Jipher Reference Information

Supported Algorithm Strings 8-1
Supported Non-FIPS 140 Allowed Algorithms 8-11
Keysize Restrictions 8-12
KeyGenerator 8-12
KeyPairGenerator 8-12
AlgorithmParameterGenerator 8-13
Supported Elliptic Curve Names 8-14
Default Diffie-Hellman Parameters 8-14
Default Digital Signature Algorithm Parameters 8-14

Preface

This guide shows you how to install, deploy, and configure Oracle Jipher, a Java cryptography service provider that provides FIPS 140 validated cryptography. This guide also provides technical information about available cryptographic engine class algorithms and default parameter values.

Audience

Jipher is intended for administrators who need to deploy their organization's Java applications in FIPS 140 regulated environments.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1 What is Oracle Jipher?

Oracle Jipher is a Java Cryptographic Service Provider (CSP) that packages a Federal Information Processing Standards (FIPS) 140 validated OpenSSL cryptographic module. It enables deployment of Java applications in FIPS regulated environments. Jipher makes its cryptographic services available to Java developers using the Java Cryptography Architecture (JCA).

Jipher 10.35 supports the following runtimes:

- Oracle JDK 17
- GraalVM for JDK 17
- Oracle JDK 21
- GraalVM for JDK 21

Jipher 10.35 supports the following platforms:

- Oracle Linux 9 and 8 on x86-64 and aarch64
- Red Hat Linux 9 and 8 on x86-64 and aarch64

For TLS, Jipher supports the JDK JSSE provider, in particular, the SunJSSE provider for TLSv1.2 and TLSv1.3.

Jipher, also known as JipherJCE, provides FIPS 140 cryptographic services by using the Java Native Interface (JNI) to make calls into an embedded copy of an OpenSSL FIPS module. It essentially maps Java cryptography API calls to OpenSSL API calls, which call into the OpenSSL FIPS module. See FIPS-140 in the OpenSSL documentation for more information.

Jipher is distributed as a JAR file. This JAR file embeds two copies of the OpenSSL FIPS module for each supported platform:

- A version of the OpenSSL FIPS module built from source code that has been tested by an accredited Cryptographic and Security Testing Laboratory (CSTL), validated by the Cryptographic Module Validation Program (CMVP), and issued a FIPS 140 validation certificate
- A version of the OpenSSL FIPS module built from the FIPS 140 compliant baseline source code with additional security patches applied, which is used by default

> **Note:**
>
> By default, Jipher prioritizes security over compliance by using the OpenSSL FIPS module with the additional security patches. However, you can configure Jipher so that a certified OpenSSL FIPS module would be used in compliance with the Federal Information Security Modernization Act (FISMA). See Configuring Jipher Through System Properties.

The Java Cryptographic Architecture (JCA), Engine Classes, and Providers

The JCA is a framework for working with cryptographic services, such as digital signature algorithms, message digest algorithms, and key conversion services.

The JCA defines classes that provide the functionality of these cryptographic services. These classes are called *engine classes*. An engine class provides the interface to a specific type of cryptographic service, independent of a particular algorithm or provider.

The JCA includes both *cryptographic engine classes*, which support cryptographic algorithms and *non-cryptographic engine classes*, which support non-cryptographic algorithms.

Cryptographic engine classes that support cryptographic algorithms provide one of the following:

- Cryptographic operations, for example, encryption, digital signatures, and message digests; these engine classes include `Cipher`, `Mac`, `KEM`, `KeyAgreement`, `MessageDigest`, and `Signature`
- Generators or converters of cryptographic material such as keys and algorithm parameters; these engine classes include `KeyGenerator`, `KeyFactory`, `SecretKeyFactory`, `SecureRandom`, `AlgorithmParameters`, and `AlgorithmParameterGenerator`

Non-cryptographic engine classes that support non-cryptographic algorithms provide one of the following:

- Objects, such as keystores and certificates, that encapsulate cryptographic data and can be used at higher layers of abstraction; these engine classes include `KeyStore`, `CertificateFactory`, `CertPathBuilder`, `CertPathValidator`, and `CertStore`
- Protocols, such as TLS, and other higher level functionality that employ cryptographic operations and the objects that encapsulate cryptographic data; these engine classes include `SSLContext`, `KeyManagerFactory`, and `TrustManagerFactory`

A CSP, which is used interchangeably with the term "provider," is a package or set of packages that implement one or more cryptographic services. To use the JCA, an application requests a particular type of object, such as a `MessageDigest`, and a particular algorithm or service, such as the SHA-256 algorithm, and gets an implementation from one of the installed providers. For example, the following statement requests a SHA-256 message digest from an installed provider:

```
md = MessageDigest.getInstance("SHA-256");
```

Alternatively, the program can request objects from a specific provider. For example, the following statement requests a SHA-256 message digest from Jipher:

```
md = MessageDigest.getInstance("SHA-256", "JipherJCE");
```

See Java Cryptography Architecture (JCA) Reference Guide in *Java Platform, Standard Edition Security Developer's Guide* for more information.

FIPS 140

The Federal Information Processing Standards (FIPS) of the United States are a set of publicly announced standards published by the National Institute of Standards and Technology (NIST). The FIPS 140 standards detail security requirements for cryptographic modules. Accredited Cryptographic and Security Testing Laboratories (CSTLs) test that cryptographic modules meet FIPS 140 security requirements. Modules that meet these security requirements are issued a Cryptographic Module Validation Program (CMVP) certificate.

Organizations that must comply with the Federal Information Security Modernization Act (FISMA) include state agencies and private sector companies with government contracts. FISMA mandates the use of FIPS 140 approved or allowed cryptography provided by cryptographic modules with CMVP certificates.

You don't have to submit Jipher for testing to a CSTL nor do you need to acquire a CMVP certificate.

Jipher Artifacts

Jipher is packaged in a `.tar.gz` file named `jipher-10.35.tar.gz`. It contains the Jipher JAR file, which is named `jipher-jce-10.35-se.jar`.

This JAR file includes the following:

- A single instance of the Jipher provider classes
- For each supported platform:
  - The Jipher native library, which provides access to OpenSSL functionality through the JNI
  - Two OpenSSL FIPS modules:
    - A version of the OpenSSL FIPS module built from source code that has been tested by a CSTL, validated by the CMVP, and issued a FIPS 140 validation certificate
    - A version of the OpenSSL FIPS module built from the FIPS 140 compliant baseline source code with additional security patches applied, which is used by default
  - An OpenSSL configuration file

Independence from and Coexistence with Other Instances of OpenSSL

Jipher explicitly loads the OpenSSL native code libraries embedded in the `jipher-jce-10.35-se.jar` JAR file. Jipher is therefore independent of any other instance of OpenSSL present on the system, including any instance that is part of the operating system distribution. The embedded libraries do not export any symbols from the OpenSSL library, so there will be no symbol clashes in a process that uses Jipher and also loads another instance of OpenSSL.

2 Downloading and Deploying Jipher

Downloading Jipher

Download Jipher from Java Tools and Resources.

Jipher Deployment

Extract the Jipher JAR file from `jipher-10.35.tar.gz`. Add the Jipher JAR file, `jipher-jce-10.35-se.jar`, to your class path or your module path. When placed on the module path, it will be observable as the module `com.oracle.jipher`.

> **Note:**
>
> The JCA authenticates a security provider by verifying the JAR file's signature at runtime. See Registering the SUN and SunRsaSign Providers for JAR File Signature Verification. Consequently, you cannot extract the contents of the Jipher JAR file and incorporate them into a JAR with dependencies (a JAR file that contains not only a Java application but includes its dependencies) to simplify deployment.

At runtime, the native libraries appropriate for the platform on which Jipher is deployed are automatically extracted from the Jipher JAR file to a temporary directory. By default, the value of the `java.io.tmpdir` system property is used as a parent directory of this temporary directory. Alternatively, you can specify this parent directory by setting the value of the `jipher.user.dir` system property. See Configuring Jipher Through System Properties.

Check the Version of Your Jipher Deployment

The Jipher provider class is `com.oracle.jipher.provider.JipherJCE`. It contains a `main` method that prints version information to standard output.

If you run this method and it prints version information without any errors, then the Jipher native libraries have been successfully loaded:

```
$ java -cp jipher-jce-10.35-se.jar com.oracle.jipher.provider.JipherJCE
JipherJCE Provider 10.35[OpenSSL 3.4.1 11 Feb 2025] (implements AES,
DESede, Diffie-Hellman, DSA, ECDSA, ECDH, HMAC, PBKDF2, RSA, SHA-1,
SHA-2, SHA-3)
```

3

Configuring an Application to Achieve FIPS 140 Compliance with Jipher

To achieve FIPS 140 compliance, your applications must use only FIPS-approved or NIST-recommended cryptography provided by CMVP-certified cryptographic modules. However, none of the providers included in the JDK are CMVP-certified.

Configure your application to achieve FIPS 140 compliance with Jipher by following these following these steps:

> **Note:**
>
> If you haven't already done so, download and deploy Jipher as described in Downloading and Deploying Jipher.

1. Register Jipher and other security providers required by the JCA and your application as described in Provider Registration:
   a. Registering Jipher as the Most Preferred Provider: It's recommended that you register Jipher as the most preferred provider, in position 1 in the list of security providers.
   b. Registering the SUN and SunRsaSign Providers for JAR File Signature Verification: The JCA requires the SUN and SunRsaSign providers for JAR file signature verification. The JDK requires this when a provider is first used to perform an encryption operation.

   > **Note:**
   >
   > The JCA requires a non-FIPS 140 allowed algorithm, the MD5withRSA `Signature` algorithm, as a prerequisite for JAR file signature verification. The SunRsaSign provider provides it.
   > With regards to FIPS 140 compliance, the MD5withRSA `Signature` algorithm is used internally for a non-security-relevant purpose and is acceptable in this instance.

   c. Registering Other Security Providers: This section lists non-cryptographic and cryptographic engine class algorithms not supported by Jipher, which your application might require, and some guidance about registering the security providers that provide these algorithms.

   You can register a provider included in the JDK as long as it's not used to provide a cryptographic engine class algorithm. (See The Java Cryptographic Architecture (JCA), Engine Classes, and Providers.) However, it can be difficult to ensure that registering one will not cause it to be used to provide a cryptographic engine class algorithm. For instance, third-party dependencies and providers themselves can use other registered providers to provide cryptographic engine class algorithms. It's therefore good practice to only register providers required by your application. One option is to register a provider and then unregister it once your application no longer needs it. See Registering the SUN and SunRsaSign Providers for JAR File Signature Verification for an example.

2. If you're using the SunJSSE provider, which enables secure Internet communications by providing implementations of the SSL, TLS, and DTLS protocols, then follow the steps described in Configure the SunJSSE Provider to Select Jipher to Provide Cryptography. The SunJSSE provider doesn't provide its own cryptographic engine class algorithms for these protocols. Instead, it uses the JCA to request implementations of cryptographic engine class algorithms from other providers. These steps ensure that the SunJSSE provider uses cryptographic engine class algorithms provided by Jipher and no other provider.
3. In your application code, ensure that Jipher is the only security provider used to provide cryptography. Follow the guidance in these sections:
   - Explicitly Selecting Jipher to Provide Cryptography
   - Implicitly Selecting Jipher to Provide Cryptography

   > **Note:**
   >
   > If your application, as opposed to the JCA as described in the previous note, requires a non-FIPS 140 allowed algorithm, then to achieve FIPS 140 compliance, you must change your application to no longer use that non-FIPS 140 allowed algorithm.

4. If required, configure some of Jipher's features through system properties. See Configuring Jipher Through System Properties.

Provider Registration

To use a security provider that's not included in the JDK, such as Jipher, you must register it so that the JCA can access its security services. You can register a provider statically or dynamically.

Registering Jipher as the Most Preferred Provider

It's recommended that you register Jipher as the most preferred provider, in position 1 in the list of security providers.

When an application requests an instance of an algorithm, without specifying which provider should provide the implementation, the JCA searches the list of registered providers in preference order. The JCA returns the implementation from the first provider supplying the algorithm. Registering Jipher as the most preferred provider ensures that the JCA will select it to provide certified implementations of algorithms allowed by FIPS 140 standards.

> **Note:**
>
> Registering Jipher as the most preferred provider does not guarantee that it will be selected to provide an implementation of an algorithm. The following are some reasons why Jipher might not be selected:
>
> - The algorithm is not approved or allowed by FIPS 140 standards.
> - The application wants to initialize the algorithm with a parameter not allowed by FIPS 140 standards because, for example, the key size is too weak or the algorithm parameters are not approved.
> - The application wants to use the algorithm for an operation not allowed by FIPS 140 standards, for example, using SHA-1 to digest data to be signed or using RSA (with PKCS #1 padding) for key transport.
> - The algorithm is approved or allowed by FIPS 140 standards but Jipher doesn't provide an implementation of it.
>
>   For example, the SUN provider provides implementations of algorithms that are approved or allowed by FIPS 140 standards that Jipher doesn't provide. These include:
>
>   - Truncated digests such as SHA-512/224, SHA-512/256, SHA512/224withDSA, and SHA512/256withDSA
>   - `Signature` algorithms that use SHA3, which include SHA3-224withDSA, SHA3-256withDSA, SHA3-384withDSA, and SHA3-512withDSA
>
>   As a result, the SUN provider (or another registered provider that doesn't provide FIPS 140 validated cryptography) should be selected to provide an implementation of an algorithm that is approved or allowed by FIPS 140 standards even though Jipher is registered as the most preferred provider.
>
>   See Registering Other Security Providers for a list of algorithms that the SUN provider supports that aren't supported by Jipher.

You register a provider like Jipher in the following ways:

- Statically by specifying it in the list of registered providers in the `java.security` file
- Dynamically by calling either the `addProvider` or `insertProviderAt` method in the `java.security.Security` class in your application code

See Step 8.1: Configure the Provider in How to Implement a Provider in the Java Cryptography Architecture in *Java Platform, Standard Edition Security Developer's Guide* for steps on how to do this.

> **Note:**
>
> It might be more fragile to register Jipher statically than dynamically. The following describes how Jipher could be configured incorrectly and result in Jipher not being used as expected. Of greatest concern is that this consequence might not be readily apparent. Your application still works and doesn't report any errors, but it's not using Jipher or FIPS 140 certified cryptography.
>
> - Suppose the Jipher provider classes are not available to a JVM instance because, for example, the `jipher-jce-10.35-se.jar` JAR file is not in the class path. As a result, the JVM instance will silently fail to register Jipher when configured to statically register it. The JVM instance will behave as though it wasn't configured to statically register Jipher. This will in turn impact your application's FIPS 140 compliance.
> - If the JDK's `java.security` file has been edited to statically register Jipher, then it might need to be re-edited if the JDK was reinstalled.

The following example is an excerpt from the `java.security` file. It registers Jipher in position 1 by specifying its provider class, `com.oracle.jipher.provider.JipherJCE`. It also registers the SUN provider in position 2, SunRsaSign in position 3, and SunJSSE in position 4:

```
security.provider.1=JipherJCE
security.provider.2=SUN
security.provider.3=SunRsaSign
security.provider.4=SunJSSE
# Other registered providers follow...
```

> **Note:**
>
> The list of statically registered providers is often longer. A more complete list better ensures that a requested algorithm that is not supported by Jipher will be provided by another provider.

In this example, because Jipher is registered at position 1, the JCA will retrieve cryptographic engine class algorithm implementations from it first. If the JCA can't retrieve an implementation of a requested algorithm from Jipher, it will attempt to retrieve an implementation from one of the other registered providers. For example a request for the MD5 message digest algorithm, which is not a FIPS 140 allowed algorithm, would result in the JCA retrieving the implementation provided by the SUN provider.

Registering the SUN and SunRsaSign Providers for JAR File Signature Verification

You must register the SUN and SunRsaSign providers because they provide algorithms, which Jipher doesn't provide, that are required for JAR file signature verification.

Security providers, like Jipher, that provide algorithms for engine classes from the `javax.crypto` package must be in a signed JAR file. The verification of a JAR file's signature occurs when getting an algorithm instance from a class in the `javax.crypto` package. If JAR file signature verification fails, then the provider is not selected to provide the implementation.

JAR file signature verification requires two algorithms that Jipher doesn't provide:

- The X.509 `CertificateFactory` algorithm provided by the SUN provider
- The MD5withRSA `Signature` algorithm provided by the SunRsaSign provider

These algorithms must be provided by a registered provider prior to verifying Jipher's JAR file signature. Do this by registering the SUN and SunRsaSign providers.

If you want to minimize the number of registered providers in your application, you can statically register the providers that are required for JAR file signature verification, then dynamically unregister them once Jipher's JAR file signature is verified. The following example demonstrates this:

```
static {
    // Dynamically register the "JipherJCE" provider if it has not already
been statically registered.
    if (Security.getProvider("JipherJCE") == null)
Security.insertProviderAt(new JipherJCE(), 1);

    // Trigger the javax.crypto.JarVerifier's certificate verification self-
test.
    // This assumes that the "SUN" and "SunRsaSign" providers have been
statically registered.
    Cipher.getInstance("AES", "JipherJCE");

    // Dynamically unregister the "SUN" and "SunRsaSign" providers
    Security.removeProvider("SUN");
    Security.removeProvider("SunRsaSign");
}
```

Registering Other Security Providers

As described in The Java Cryptographic Architecture (JCA), Engine Classes, and Providers, some providers included in the JDK provide both cryptographic engine class algorithms and non-cryptographic engine class algorithms. However, Jipher doesn't provide any non-cryptographic engine class algorithms.

In addition, the SUN and SunRsaSign providers are needed for provider JAR file signature verification. The JDK requires this when a provider is first used to perform an encryption operation. See Registering the SUN and SunRsaSign Providers for JAR File Signature Verification.

However, the fewer security providers that are registered, the lower the chance that a provider other than Jipher will be selected to provide a cryptographic engine class algorithm that Jipher doesn't support (due to FIPS 140 restrictions) or, in the case of delayed provider selection (see Delayed Provider Selection), a cryptographic engine class algorithm that Jipher does support initialized with a `Key` it does not support (due to FIPS 140 restrictions).

The SUN provider is optionally required to provide support for the following non-cryptographic engine class algorithms:

- Loading certificates and building/validating certificate paths:
  - `X.509` `CertificateFactory` algorithm

  - PKIX CertPathBuilder and CertPathValidator algorithms
  - Collection CertStore algorithm
- Loading keystores and truststores:
  - PKCS12 KeyStore algorithm

The SunJSSE provider is optionally required to provide TLS support through these non-cryptographic engine class algorithms:

- PKIX KeyManagerFactory and TrustManagerFactory algorithms
- All the SSLContext algorithms

Note that the SUN and SunRsaSun providers support cryptographic engine class algorithms that are supported by Jipher. Consequently, if you register these providers, it's important that Jipher is registered with a higher preference than them as described in Registering Jipher as the Most Preferred Provider.

The SUN provider supports the following cryptographic engine class algorithms that are not supported by Jipher:

- KeyFactory:
  - HSS/LMS
- MessageDigest:
  - MD2
  - MD5
  - SHA-512/224
  - SHA-512/256
- Signature:
  - HSS/LMS
  - NONEwithDSAinP1363Format
  - SHA1withDSAinP1363Format
  - SHA224withDSAinP1363Format
  - SHA256withDSAinP1363Format
  - SHA384withDSAinP1363Format
  - SHA512withDSAinP1363Format
  - SHA3-224withDSA
  - SHA3-256withDSA
  - SHA3-384withDSA
  - SHA3-512withDSA
  - SHA3-224withDSAinP1363Format
  - SHA3-256withDSAinP1363Format
  - SHA3-384withDSAinP1363Format
  - SHA3-512withDSAinP1363Format

The SunRsaSign provider supports the following Signature engine class algorithms that are not supported by Jipher:

- MD2withRSA
- MD5withRSA
- SHA512/224withRSA
- SHA512/256withRSA
- SHA3-224withRSA
- SHA3-256withRSA
- SHA3-384withRSA
- SHA3-512withRSA

The SunJSSE provider does not support any cryptographic engine class algorithms not supported by Jipher.

Delayed Provider Selection

Java cryptographic API classes that generate, import, or can be initialized with a key (`KeyGenerator`, `KeyPairGenerator`, `KeyFactory`, `SecretKeyFactory`, `Cipher`, `KeyAgreement`, `Mac`, and `Signature`) support delayed provider selection. A `getInstance(String algorithm)` method call compiles a preference ordered list of providers that support the algorithm. Later, when the `init` method is called, the first provider in the compiled preference ordered list of providers that also supports the key (with regards to size and format) is selected.

Configure the SunJSSE Provider to Select Jipher to Provide Cryptography

The SunJSSE provider enables secure Internet communications by providing implementations of the SSL, TLS, and DTLS protocols. It doesn't provide its own cryptographic engine class algorithms for these protocols. Instead, it uses the JCA to request implementations of cryptographic engine class algorithms from other providers. Follow these configuration steps to ensure that the SunJSSE provider uses cryptographic engine class algorithms provided by Jipher and no other provider:

1. Permit Access to Internal JDK API Classes: This ensures that Jipher is allowed to provide cryptography requested by the SunJSSE provider.
2. Registering Jipher as the Most Preferred Provider: This ensures that Jipher will be chosen ahead of any other providers to provide an implementation of any algorithm it supports. Also, register the SUN and SunRsaSign providers as described in Registering the SUN and SunRsaSign Providers for JAR File Signature Verification.
3. Limit Protocol Versions, Cipher Suites, Key Material, and Named Groups: This ensures that the SunJSSE provider never requests cryptography that Jipher does not support.

These steps combined ensure that the SunJSSE provider never uses any cryptography provided by a non-FIPS 140 certified registered provider.

> **Note:**
>
> If the following is true about your environment, then you only have to follow the steps Permit Access to Internal JDK API Classes and Configure SSLContext with Key Material That Provides at Least 112-Bits of Security:
>
> - You're using JDK 21.
> - It uses a standard JDK security property configuration where:
>   - The `jdk.certpath.disabledAlgorithms` security property specifies `MD2` and `MD5`.
>   - The `jdk.tls.disabledAlgorithms` security property specifies `SSLv3`, `TLSv1`, `TLSv1.1`, and `MD5withRSA`.
> - Jipher is registered as the most preferred provider and the only other registered providers are SUN, SunRsaSign, and SunJSSE. See Registering Jipher as the Most Preferred Provider.

Permit Access to Internal JDK API Classes

When a security provider provides the cryptography required by the SunJSSE to provide TLSv1.2, it must access internal JDK API classes.

If Jipher attempts to access them, an `IllegalAccessException` is thrown. To prevent this from happening, add one of the following command-line options when you run your Java application depending on where you specified the Jipher JAR file

- On the class path:

  ```
  --add-exports=java.base/sun.security.internal.spec=ALL-UNNAMED
  ```

- On the module path:

  ```
  --add-exports=java.base/sun.security.internal.spec=com.oracle.jipher
  ```

Limit Protocol Versions, Cipher Suites, Key Material, and Named Groups

This section describes which SunJSSE parameters and properties to limit to the same levels and values as Jipher. This ensures that the SunJSSE provider never requests cryptography that Jipher does not support.

Ensure That Clients and Servers Only Use TLS 1.2 and TLS 1.3

Ensure that the security property `jdk.tls.disabledAlgorithms` includes the values `SSLv3`, `TLSv1`, and `TLSv1.1`.

In addition, ensure that the system properties `jdk.tls.server.protocols` and `jdk.tls.client.protocols` contain the values `TLSv1.2` and `TLSv1.3`. See Customizing JSSE in *Java Platform, Standard Edition Security Developer's Guide* for more information about these system properties and other properties you can specify to customize JSSE.

In your code, call `SSLContext.getInstance("TLS")` instead of supplying specific TLS protocol versions.

> **Note:**
>
> Calling `SSLContext.getInstance("TLS1.3")` sets the ceiling for protocol versions. As a result, this statement returns a context that allows TLS 1.0 and TLS 1.1 protocol versions.

Ensure TLS Cipher Suites Use FIPS 140 Algorithms from Jipher

Depending on which TLS protocol version you want to support (TLS 1.2 or TLS 1.3), specify the cipher suites that only require FIPS 140 allowed cryptography provided by Jipher as a comma-separated list for the system properties `jdk.tls.client.cipherSuites` and `jdk.tls.server.cipherSuites`. See Specifying Default Enabled Cipher Suites in *Java Platform, Standard Edition Security Developer's Guide* for more information about these system properties.

See The SunJSSE Provider in *Java Platform, Standard Edition Security Developer's Guide* for a complete list of supported cipher suites.

TLS 1.2 cipher suites are expressed as follows:

```
TLS_<key exchange algorithm>_<authentication algorithm>_WITH_<cipher algorithm>_<cipher strength>_<cipher mode>_<HASH or MAC>
```

TLS 1.3 cipher suites are expressed as follows:

```
TLS_<cipher algorithm>_<cipher strength>_<cipher mode>_<HASH or MAC>
```

The following table lists the values within a cipher suite's name that are supported by Jipher:

Table 3-1 Values Within a Cipher Suite's Name Supported by Jipher

Component	Supported	Supported but Not Recommended
Key exchange algorithm	ECDHE, DHE	DH
Authentication algorithm	ECDSA, RSA	DSS
Cipher algorithm	AES	—
Cipher strength	256, 128	—
Cipher mode	GCM	CBC
HASH or MAC	SHA384, SHA256	SHA

Cipher suites that use RSA for key exchange and authentication are expressed as follows:

```
TLS_RSA_WITH_<cipher algorithm>_<cipher strength>_<cipher mode>_<HASH or MAC>
```

These cipher suites are not supported.

See "Appendix D—RSA Key Transport" in NIST SP 800-52 Rev. 2: Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations for the reasons why these cipher suites are not supported. In short, it is because RSA key transport as used in TLS versions 1.0 through 1.2 is implemented using the PKCS #1 v1.5 padding scheme.

Configure SSLContext with Key Material That Provides at Least 112-Bits of Security

Key managers are responsible for managing the key material which is used to authenticate the local TLS endpoint to its peer. Configure your key manager with key material with the following parameters so that the key material provides at least 112-bits of security:

- RSA: Modulus >= 2048
- DSA: (L, N) = (2048, 224), (2048, 256) or (3072, 256)
- EC: secp224r1, secp256r1, secp384r1, or secp521r1

See the section KeyManagerFactory Class in *Java Platform, Standard Edition Security Developer's Guide*.

> **Note:**
>
> The system property `jdk.tls.disabledAlgorithms` has no impact on key size used by the local TLS endpoint to generate a digital signature to authenticate itself to its peer.
> While `jdk.tls.disabledAlgorithms=DSA`, disables DSA (disables DSS cipher suites), `jdk.tls.disabledAlgorithms=DSA keySize < 2048` neither disables DSS cipher suites nor does it prevent the local TLS endpoint from using a DSA private key of less than 2048 bits when generating a signature to authenticate itself to its peer.

Configure the Size of Ephemeral Diffie-Hellman Keys

Set the system property `jdk.tls.ephemeralDHKeySize` to `2048`.

Ensure the JSSE Only Uses FIPS 140 Approved Named Groups

FIPS 140 compliance requires the use of approved elliptic curves and safe-prime groups for key establishment as documented in "Appendix D: Approved ECC Curves and FFC Safe-prime Groups" in NIST SP 800-56A Rev. 3: Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography.

Set the system property `jdk.tls.namedGroups` to only allow the following:

- FIPS 140 approved named curves:
  - `secp256r1`
  - `secp384r1`
  - `secp521r1`
- FIPS 140 approved FFC safe-prime Groups:
  - `ffdhe2048`
  - `ffdhe3072`
  - `ffdhe4096`

Explicitly Selecting Jipher to Provide Cryptography

When an application calls `getInstance` from an engine class such as `MessageDigest`, `Signature`, `KeyFactory`, `KeyPairGenerator`, or `Cipher` to acquire an instance of a cryptographic engine class algorithm, it can explicitly specify which provider should provide the algorithm by calling one of the following:

- `getInstance(String algorithm, String provider)`
- `getInstance(String algorithm, Provider provider)`

One way to ensure that Jipher is the only security provider used to provide cryptography is to explicitly specify that Jipher provide the instance of the algorithm in all `getInstance` method calls used to acquire an instance of a cryptographic engine class algorithm, for example:

```
Cipher.getInstance("AES", "JipherJCE");
```

In practice, this is often infeasible because of the following:

- Applications often need to use other security providers to provide higher level functionality, and these providers will internally make `getInstance(String algorithm)` calls to acquire the cryptographic engine class algorithms they need to deliver the higher level functionality they provide. For example the PKCS12 `KeyStore` algorithm provided by the SUN provider internally uses other security providers to provide the cryptographic engine class algorithms it uses to secure the key store.
- Applications often use third-party libraries that internally make `getInstance(String algorithm)` calls to acquire cryptographic engine class algorithms they need to deliver the functionality they provide.

Consequently, if your application, or one of its dependencies, calls `getInstance(String algorithm)` from an engine class, then configure the JVM so that Jipher is implicitly selected to provide the algorithm. See Implicitly Selecting Jipher to Provide Cryptography.

Implicitly Selecting Jipher to Provide Cryptography

To implicitly select Jipher whenever an application or one of its dependencies calls an engine class's `getInstance(String)` method, follow these steps:

1. Register Jipher as the most preferred provider. See Provider Registration.
2. Configure your application and its dependencies to only request cryptographic engine class algorithms that Jipher supports. For example, if you're using the SunJSSE provider to provide secure Internet communications, then follow the steps described in Configure the SunJSSE Provider to Select Jipher to Provide Cryptography. If your application requires a non-FIPS 140 allowed algorithm, then to achieve FIPS 140 compliance, you must change your application to no longer use that non-FIPS 140 allowed algorithm.

Configuring Jipher Through System Properties

You can configure some of the features of Jipher through the following system properties:

Table 3-2 Jipher System Properties

System Property	Description	Default Value
`java.security.debug`	Standard Java system property. If the value includes `jipher` or `all`, then debug logging through `System.err` is enabled within Jipher. This prints debugging information, including the library loading steps that are performed when Jipher is first used.	*None*
`java.io.tmpdir`	Standard Java system property. This system property specifies the location of temporary files. Jipher uses this value as the default location in which to create temporary directories for storing native libraries. See `jipher.user.dir`.	Typically `/tmp` on Linux operating systems
`jipher.user.dir`	Specifies the path location that Jipher uses to create temporary directories to store library files. The user running the JVM process must have write access to this directory when using Jipher. In addition, this directory cannot be on a file system mounted with the `noexec` option. **Note:** Sometimes the `/tmp` directory is mounted with the `noexec` option. If this is the case, then set the `jipher.user.dir` system property to another directory that has write access and is on a file system that has not been mounted with the `noexec` option.	The value of `java.io.tmpdir`, typically `/tmp` on Linux operating systems
`jipher.fips.enforcement`	Specifies the FIPS 140 enforcement policy to be applied at the Java security provider layer. It can have one of the following values: • `FIPS`: Approved algorithms and key lengths are permitted in accordance with how they are used. Legacy use is permitted. • `FIPS_STRICT`: Only algorithms and key lengths with Acceptable approval status are permitted. • `NONE`: No additional enforcement is performed; any enforcement implemented in OpenSSL is still applied. This policy is intended to be used during preproduction debugging. You may encounter behavior that differs from the other policies such as: – A different exception being thrown – A delay in an exception being thrown; for example, an exception may be thrown when a signature object is used instead of when it is created or initialized *Acceptable* is used by NIST to mean that the algorithm and key length is safe to use. *Legacy use* means that the algorithm or key length may be used only to process already protected information, for example, to decrypt ciphertext data or to verify a digital signature. See NIST SP 800-131A Rev. 2: Transitioning the Use of Cryptographic Algorithms and Key Lengths and FIPS 140 Enforcement Policy for more information.	`FIPS`

Table 3-2 (Cont.) Jipher System Properties

System Property	Description	Default Value
`jipher.fips.deactivateSecurityPatches`	Specifies whether Jipher prioritizes security (`false`) or compliance (`true`). The Jipher JAR file embeds two copies of the OpenSSL FIPS module for each supported platform: • A version of the OpenSSL FIPS module built from source code that has been tested by a CSTL, validated by the CMVP, and issued a FIPS 140 validation certificate • A version of the OpenSSL FIPS module built from the FIPS 140 compliant baseline source code with additional security patches applied, which is used by default The system property `jipher.fips.deactivateSecurityPatches` can have one of the following values: • `false`: Jipher prioritizes security over compliance. It uses the OpenSSL FIPS module with the additional security patches to implement all the cryptography it provides. • `true`: Jipher prioritizes compliance over security. It uses the OpenSSL FIPS module without the additional security patches. **Caution:** Setting `jipher.fips.deactivate.security.patches` to `true` incurs the serious risk that security vulnerabilities, published as Common Vulnerabilities and Exposures (CVE) in the public domain, will become exploitable in your deployment. However, any change to the OpenSSL FIPS module following FIPS 140 validation renders it non-compliant, so Jipher's default operation, setting `jipher.fips.deactivate.security.patches` to `false`, is not strictly FIPS 140 compliant with FISMA. Some users prefer to be strictly FIPS 140 compliant and use the CSTL-tested and CMVP-validated OpenSSL FIPS module that has been issued a FIPS 140 validation certificate. Others, like Oracle, use a recent baseline of the OpenSSL FIPS module that is FIPS 140 compliant, add security patches, and disclose this choice to customers and auditors. Regardless of whether you choose to be strictly FIPS 140 compliant or use the OpenSSL FIPS module with the additional security patches, Oracle recommends disclosing this choice to customers and auditors.	`false`

FIPS 140 Enforcement Policy

Jipher can perform enforcement of FIPS 140 algorithm usage and key sizes according to how the algorithm and key is being used.

You can specify a FIPS 140 enforcement policy with the `jipher.fips.enforcement` system property. See Configuring Jipher Through System Properties. The default policy, `FIPS`, permits legacy use. This means that an algorithm or key length with "legacy use" status may be used, but only to process already protected information, for example, to decrypt ciphertext data or to verify a digital signature. The `FIPS_STRICT` policy only permits algorithms and key lengths with Acceptable approval status. This is more restrictive though likely to be more stable over time. See NIST SP 800-131A Rev. 2: Transitioning the Use of Cryptographic Algorithms and Key Lengths for more information..

In general, the default `FIPS` policy is similar to the `FIPS_STRICT` policy except that the `FIPS_STRICT` policy requires longer key lengths for digital signature verification and MACs for various algorithms. The following table highlights these differences in **bold**.

Table 3-3 Minimum Key Lengths of the Default FIPS Policy and the FIPS_STRICT Policy

Rule	FIPS Policy	FIPS_STRICT Policy
AES key bits	>= 128 for symmetric encryption and decryption	>= 128 for symmetric encryption and decryption
DESede key bits	Not supported for symmetric encryption = 192 for symmetric decryption	Not supported for symmetric encryption **and decryption**
DH key bits	>= 2048 for key generation and key agreement	>= 2048 for key generation and key agreement
DSA parameter bits	(prime size, sub-prime size) is one of (2048, 224), (2048, 256) or (3072, 256) for key generation and signature generation	(prime size, sub-prime size) is one of (2048, 224), (2048, 256) or (3072, 256) for key generation and signature generation **and verification**
DSA parameter bits	prime size > 512 bits for signature verification	*See the previous table cell in this column*
EC curve bits	>= 224 for key generation, signature generation and key agreement	>= 224 for key generation, signature generation, **verification**, and key agreement
EC curve bits	>= 160 for signature verification	*See the previous table cell in this column*
HMAC key bits	>= 0 for MAC	>= **112** for MAC
Message digest algorithm	!= SHA-1 for signature generation	!= SHA-1 for signature generation **and verification**
RSA key bits	>= 2048 for asymmetric encryption and decryption key generation and signature generation	>= 2048 for asymmetric encryption and decryption key generation and signature generation **and verification**
RSA key bits	>= 1024 for signature verification.	*See the previous table cell in this column*

Table 3-3 (Cont.) Minimum Key Lengths of the Default FIPS Policy and the FIPS_STRICT Policy

Rule	FIPS Policy	FIPS_STRICT Policy
RSA padding	!= PKCS1 for asymmetric encryption	!= PKCS1 for asymmetric encryption

4

Applying NIST Guidelines for TLS

NIST SP 800-52 Rev. 2: Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations specify that TLS servers should use minimum key sizes when using certain algorithms and must perform revocation checking of the client certificate when client authentication is used.

Configure Minimum Key Sizes for Certificate Path Validation

Update the security property `jdk.certpath.disabledAlgorithms` to add the following restrictions:

- `RSA keySize < 2048`
- `EC keySize < 256`

See Disabled and Restricted Cryptographic Algorithms in *Java Platform, Standard Edition Security Developer's Guide*.

Enable Revocation Checking

A certificate is a digitally signed statement, typically issued by a Certificate Authority (CA), vouching for the identity and public key of an entity. Certificates used in TLS can be revoked by the issuing CA if there is reason to believe that a certificate is compromised. NIST guidelines specify that servers must perform revocation checking of the client certificate when client authentication is used. In addition, the server must retrieve revocation information though the Online Certificate Status Protocol (OCSP) or Certificate Revocation Lists (CRLs).

See PKIX TrustManager Support and Client-Driven OCSP and OCSP Stapling in *Java Platform, Standard Edition Security Developer's Guide* for more information.

Follow these steps to enable revocation checking and client-driven OCSP.

1. Set the system property `com.sun.net.ssl.checkRevocation` to `true`.
2. Set the system property `ocsp.enable` to `true`.
3. Set the system property `com.sun.security.enableCRLDP` to `true`.

5 Jipher Performance Optimizations

KeyManager Selection

Only select NewSunX509, an `X509KeyManager` implementation in the SunJSSE provider, if your application explicitly requires support for multiple and dynamic keystores, for example, a server that provides Server Name Indication (SNI) support. See Multiple and Dynamic Keystores in *Java Platform, Standard Edition Security Developer's Guide* for more information.

Elliptic Curve Key Pair Generation

The performance of the elliptic curve key pair generation algorithm provided by Jipher varies depending on the elliptic curve.

Among the curves supported by Jipher, the National Security Agency (NSA) recommends the use of the secp384r1 curve; refer to the Commercial National Security Algorithm (CSNA) Suite 1.0. However, the secp256r1 curve also supported by Jipher provides better performance. See Supported Elliptic Curve Names.

6 Secure Coding Guidance

Avoiding Unnecessary In-Memory Buffering of Plaintext

The `Cipher` methods `update` and `doFinal` support data streaming. However, cipher transformations that use an AES KeyWrap algorithm (defined in RFC 3394: Advanced Encryption Standard (AES) Key Wrap Algorithm) such as AESWrap, AESWrapPad, AES/KW/NoPadding, and AES/KWP/NoPadding don't lend themselves to data streaming because all input data must be available before any of the input data can be fully processed. Consequently, if an AESWrap transform `Cipher` object is initialized with the `ENCRYPT_MODE` operation, any plaintext passed to an `update` method is copied into an internal buffer so that it may be later processed during a subsequent `doFinal` method call. The `Cipher` object's internal plaintext buffer is zeroed and freed when `doFinal` is invoked or when the `Cipher` object is garbage collected. Applications that want to avoid plaintext being buffered by an AESWrap transform `Cipher` object should avoid calling `update`. For example, consider the following code:

```
Cipher wrapper = Cipher.getInstance("AESWrap");
wrapper.init(Cipher.ENCRYPT_MODE, secretKey);
wrapper.update(plaintext);
byte[] cipherText = wrapper.doFinal();
```

You can replace it with the following:

```
Cipher wrapper = Cipher.getInstance("AESWrap");
wrapper.init(Cipher.ENCRYPT_MODE, secretKey);
byte[] cipherText = wrapper.doFinal(plaintext);
```

7

Jipher Diagnostics

Confirming That a Java Application Is Using Jipher

If you set the system property `java.security.debug` to `provider`, the JVM will print trace messages (typically to stderr) while your application is running that indicate which provider is being used to provide each service and algorithm. The output is similar to the following:

```
Provider: MessageDigest.SHA-256 algorithm from: JipherJCE
Provider: Signature.SHA256withRSA verification algorithm from: JipherJCE
Provider: KeyGenerator.SunTls12Prf algorithm from: JipherJCE
```

If you set the system property `java.security.debug` to `jipher`, Jipher will print some additional debugging information, including logging of library loading steps on first usage. The output is similar to the following:

```
jipher: Libraries found in classpath JAR, loading from jar.
jipher: Attempting to locate libraries in classpath JAR file
jipher: Found jar:file:/usr/local/lib/jipher-jce-10.35-se.jar!/libs/linux_x64/
libjipher.so
jipher: Found jar:file:/usr/local/lib/jipher-jce-10.35-se.jar!/libs/linux_x64/
patched-fips.so
jipher: Found jar:file:/usr/local/lib/jipher-jce-10.35-se.jar!/libs/linux_x64/
patched-openssl.cnf
jipher: Creating temporary directory to store libraries: /tmp/
jiphertmp-10.35-1510124983029925122
jipher: Copying jar:file:/usr/local/lib/jipher-jce-10.35-se.jar!/libs/
linux_x64/libjipher.so contents to file /tmp/
jiphertmp-10.35-1510124983029925122/libjipher.so
jipher: Copying jar:file:/usr/local/lib/jipher-jce-10.35-se.jar!/libs/
linux_x64/patched-fips.so contents to file /tmp/
jiphertmp-10.35-1510124983029925122/fips.so
jipher: Copying jar:file:/usr/local/lib/jipher-jce-10.35-se.jar!/libs/
linux_x64/patched-openssl.cnf contents to file /tmp/
jiphertmp-10.35-1510124983029925122/openssl.cnf
jipher: Loading /tmp/jiphertmp-10.35-1510124983029925122/libjipher.so...
jipher: ...Done
jipher: Configuring openssl to load FIPS module from dir: /tmp/
jiphertmp-10.35-1510124983029925122
jipher: Configuring openssl using configuration in file: /tmp/
jiphertmp-10.35-1510124983029925122/openssl.cnf
jipher: FIPS ctx: fips available = 1, default available = 1;  NULL ctx: fips
available = 0, default available = 0
jipher: Setting FIPS enforcement policy = FIPS
```

See The java.security.debug System Property in *Java Platform, Standard Edition Security Developer's Guide* for more information.

Keeping Track of Security Provider Usage with the jdk.SecurityProviderService Java Flight Recorder (JFR) Event

In JDK 20 and later, the Java Flight Recorder (JFR) event `jdk.SecurityProviderService` records the details of `java.security.Provider.getService(String type, String algorithm)` calls. This event contains the following fields:

Table 7-1 JFR Event jdk.SecurityProviderService Fields

Field Name	Description
type	Type of service
algorithm	Algorithm name
provider	Security provider

You can use the JFR event `jdk.SecurityProviderService` to confirm that a Java application is using Jipher. This JFR event is disabled by default. You can enable it through JFR configuration files or standard JFR options.

Reporting the Enforcement of FIPS 140 Restrictions

When enforcing FIPS 140 restrictions, Jipher throws an `InvalidParameterException` if directed to generate the following:

- A `SecretKey` or `KeyPair` with a security strength of less than 112 bits
- A DSA `AlgorithmParameters` or a DSA `KeyPair` using domain parameters with sizes other that those allowed by FIPS 140, which are (P=2048,Q=224), (P=2048,Q=256), and (P=3072,Q=256)
- A Diffie-Hellman `KeyPair` using domain parameters that are not a FIPS 140 approved safe prime group (see "Appendix D: Approved ECC Curves and FFC Safe-prime Groups" in NIST SP 800-56A Rev. 3: Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography)
- An elliptic curve `KeyPair` using a curve that is not a FIPS 140 approved `secp` curve (see again "Appendix D: Approved ECC Curves and FFC Safe-prime Groups").

Similarly, Jipher throws a `ProviderException` if directed to use the following:

- A `KeyPair` with a security strength of less than 80 bits to process secured data, for example, to verify a signature or decrypt cipher text
- A `SecretKey` or `KeyPair` with a security strength of less than 112 bits to secure data, for example, to generate a digital signature or encrypt plaintext
- SHA-1 to generate a signature
- A DSA `KeyPair` that does not use domain parameters allowed by FIPS 140 (listed previously)

See "Table 2: Comparable security strengths of symmetric block cipher and asymmetric-key algorithms" in NIST SP 800-57 Part 1 Rev. 5: Recommendation for Key Management: Part 1 – General for the estimated security strengths of specific algorithms and key lengths.

Reporting Misconfiguration

If Jipher can't extract the embedded native libraries to a temporary directory in the file system and load them from there into the JVM process, then it throws a `ProviderException`. This can happen if the user running the JVM process does not have permission to create the temporary directory or to execute binaries stored in the encompassing file system.

If Jipher is statically registered and a `ProviderException` is thrown when loading the native libraries, then the provider will not be registered. Other statically registered providers will still be registered.

Reporting Abnormal Operation in OpenSSL Native Code

An error condition that arises in OpenSSL native code is reported to the application through the following:

- A `java.lang.Error`, such as `java.lang.OutOfMemoryError`
- A `java.lang.RuntimeException`, either `java.lang.ArrayIndexOutOfBoundsException` or `java.lang.IllegalArgumentException`, indicating a programming error
- A `ProviderException` whose chained cause is an internal `OpenSslException` whose detail message describes the OpenSSL error stack for use in debugging and troubleshooting

8

Jipher Reference Information

Supported Algorithm Strings

The following table lists the algorithm strings and their aliases supported by Jipher. These strings are grouped by their associated engine class.

Table 8-1 Algorithm Strings Supported by Jipher

Engine	Supported Algorithm Strings and Their Aliases	Notes
SecureRandom	`DRBG` (`SHA1PRNG`, `CTRDRBG`, `CTRDRBG128`, `NativePRNG`, `NativePRNGNonBlocking`)	All aliases use the same underlying DRBG algorithm from OpenSSL
MessageDigest	`SHA-1` (`SHA`, `SHA1`, `1.3.14.3.2.26`, `OID.1.3.14.3.2.26`)	—
	`SHA-224` (`SHA224`, `2.16.840.1.101.3.4.2.4`, `OID.2.16.840.1.101.3.4.2.4`) `SHA-256` (`SHA256`, `2.16.840.1.101.3.4.2.1`, `OID.2.16.840.1.101.3.4.2.1`) `SHA-384` (`SHA384`, `2.16.840.1.101.3.4.2.2`, `OID.2.16.840.1.101.3.4.2.2`) `SHA-512` (`SHA512`, `2.16.840.1.101.3.4.2.3`, `OID.2.16.840.1.101.3.4.2.3`)	—
	`SHA3-224` (`2.16.840.1.101.3.4.2.7`, `OID.2.16.840.1.101.3.4.2.7`) `SHA3-256` (`2.16.840.1.101.3.4.2.8`, `OID.2.16.840.1.101.3.4.2.8`) `SHA3-384` (`2.16.840.1.101.3.4.2.9`, `OID.2.16.840.1.101.3.4.2.9`) `SHA3-512` (`2.16.840.1.101.3.4.2.10`, `OID.2.16.840.1.101.3.4.2.10`)	—
Cipher	`AES` (`Rijndael`, `2.16.840.1.101.3.4.1`, `OID.2.16.840.1.101.3.4.1`) `AES/CTR/NoPadding`	—

Table 8-1 (Cont.) Algorithm Strings Supported by Jipher

Engine	Supported Algorithm Strings and Their Aliases	Notes
	AES_128/ECB/NoPadding (2.16.840.1.101.3.4.1.1, OID.2.16.840.1.101.3.4.1.1) AES_192/ECB/NoPadding (2.16.840.1.101.3.4.1.21, OID.2.16.840.1.101.3.4.1.21) AES_256/ECB/NoPadding (2.16.840.1.101.3.4.1.41, OID.2.16.840.1.101.3.4.1.41) AES_128/CBC/PKCS5Padding (AES_128/CBC/PKCS7Padding, 2.16.840.1.101.3.4.1.2, OID.2.16.840.1.101.3.4.1.2) AES_192/CBC/PKCS5Padding (AES_192/CBC/PKCS7Padding, 2.16.840.1.101.3.4.1.22, OID.2.16.840.1.101.3.4.1.22) AES_256/CBC/PKCS5Padding (AES_256/CBC/PKCS7Padding, 2.16.840.1.101.3.4.1.42, OID.2.16.840.1.101.3.4.1.42) AES_128/OFB/NoPadding (2.16.840.1.101.3.4.1.3, OID.2.16.840.1.101.3.4.1.3) AES_192/OFB/NoPadding (2.16.840.1.101.3.4.1.23, OID.2.16.840.1.101.3.4.1.23) AES_256/OFB/NoPadding (2.16.840.1.101.3.4.1.43, OID.2.16.840.1.101.3.4.1.43) AES_128/CFB/NoPadding (2.16.840.1.101.3.4.1.4, OID.2.16.840.1.101.3.4.1.4) AES_192/CFB/NoPadding (2.16.840.1.101.3.4.1.24, OID.2.16.840.1.101.3.4.1.24) AES_256/CFB/NoPadding (2.16.840.1.101.3.4.1.44, OID.2.16.840.1.101.3.4.1.44) AES/GCM/NoPadding AES_128/GCM/NoPadding (2.16.840.1.101.3.4.1.6, OID.2.16.840.1.101.3.4.1.6) AES_192/GCM/NoPadding (2.16.840.1.101.3.4.1.26, OID.2.16.840.1.101.3.4.1.26) AES_256/GCM/NoPadding (2.16.840.1.101.3.4.1.46, OID.2.16.840.1.101.3.4.1.46)	—

Table 8-1 (Cont.) Algorithm Strings Supported by Jipher

Engine	Supported Algorithm Strings and Their Aliases	Notes
	DESede (TripleDES) DESede/CBC/PKCS5Padding (DESede/CBC/PKCS7Padding, OID.1.2.840.113549.3.7, 1.2.840.113549.3.7)	—
	AES/KW/NoPadding (AESWrap, AES-KW) AES_128/KW/NoPadding (AESWrap_128, 2.16.840.1.101.3.4.1.5, OID.2.16.840.1.101.3.4.1.5) AES_192/KW/NoPadding (AESWrap_192, 2.16.840.1.101.3.4.1.25, OID.2.16.840.1.101.3.4.1.25) AES_256/KW/NoPadding (AESWrap_256, 2.16.840.1.101.3.4.1.45, OID.2.16.840.1.101.3.4.1.45)	RFC 3394
	AES/KWP/NoPadding (AESWrapPad, AES-KWP) AES_128/KWP/NoPadding (AESWrapPad_128, 2.16.840.1.101.3.4.1.8, OID.2.16.840.1.101.3.4.1.8) AES_192/KWP/NoPadding, (AESWrapPad_192, 2.16.840.1.101.3.4.1.28, OID.2.16.840.1.101.3.4.1.28) AES_256/KWP/NoPadding (AESWrapPad_256, 2.16.840.1.101.3.4.1.48, OID.2.16.840.1.101.3.4.1.48)	RFC 5649
	PBEWithHmacSHA1AndAES_128 PBEWithHmacSHA224AndAES_128 PBEWithHmacSHA256AndAES_128 PBEWithHmacSHA384AndAES_128 PBEWithHmacSHA512AndAES_128 PBEWithHmacSHA1AndAES_256 PBEWithHmacSHA224AndAES_256 PBEWithHmacSHA256AndAES_256 PBEWithHmacSHA384AndAES_256 PBEWithHmacSHA512AndAES_256	PBES2 password-based cipher
	PBEWithSHA1AndDESede (1.2.840.113549.1.12.1.3, OID.1.2.840.113549.1.12.1.3)	PKCS #12 password-based encryption. The key derivation function used for this algorithm is a not a FIPS 140 allowed algorithm. This algorithm will be removed in a future release of Jipher. See Supported Non-FIPS 140 Allowed Algorithms.
	RSA/ECB/PKCS1Padding (RSA) RSA/ECB/NoPadding	—

Table 8-1 (Cont.) Algorithm Strings Supported by Jipher

Engine	Supported Algorithm Strings and Their Aliases	Notes
	RSA/ECB/OAEPPadding RSA/ECB/OAEPWithSHA-1andMGF1Padding (RSA/ECB/OAEPWithSHA1andMGF1Padding) RSA/ECB/OAEPWithSHA-224andMGF1Padding (RSA/ECB/OAEPWithSHA224andMGF1Padding) RSA/ECB/OAEPWithSHA-256andMGF1Padding (RSA/ECB/OAEPWithSHA256andMGF1Padding) RSA/ECB/OAEPWithSHA-384andMGF1Padding (RSA/ECB/OAEPWithSHA384andMGF1Padding) RSA/ECB/OAEPWithSHA-512andMGF1Padding (RSA/ECB/OAEPWithSHA512andMGF1Padding)	—
KeyFactory	RSA (1.2.840.113549.1.1, OID.1.2.840.113549.1.1, 1.2.840.113549.1.1.1, OID.1.2.840.113549.1.1.1) RSASSA-PSS (PSS, 1.2.840.113549.1.1.10, OID.1.2.840.113549.1.1.10) EC (EllipticCurve, 1.2.840.10045.2.1, OID.1.2.840.10045.2.1) DSA (1.2.840.10040.4.1, OID.1.2.840.10040.4.1, 1.3.14.3.2.12) DH (DiffieHellman, 1.2.840.113549.1.3.1, OID.1.2.840.113549.1.3.1)	—
Signature	SHA1withRSA (1.2.840.113549.1.1.5, OID.1.2.840.113549.1.1.5, 1.3.14.3.2.29, OID.1.3.14.3.2.29) SHA224withRSA (1.2.840.113549.1.1.14, OID.1.2.840.113549.1.1.14) SHA256withRSA (1.2.840.113549.1.1.11, OID.1.2.840.113549.1.1.11) SHA384withRSA (1.2.840.113549.1.1.12, OID.1.2.840.113549.1.1.12) SHA512withRSA (1.2.840.113549.1.1.13, OID.1.2.840.113549.1.1.13) NONEwithRSA	RSA with PKCS1
	RSASSA-PSS (1.2.840.113549.1.1.10, OID.1.2.840.113549.1.1.10)	—

Table 8-1 (Cont.) Algorithm Strings Supported by Jipher

Engine	Supported Algorithm Strings and Their Aliases	Notes
	SHA1withECDSA (1.2.840.10045.4.1, OID.1.2.840.10045.4.1) SHA224withECDSA (1.2.840.10045.4.3.1, OID.1.2.840.10045.4.3.1) SHA256withECDSA (1.2.840.10045.4.3.2, OID.1.2.840.10045.4.3.2) SHA384withECDSA (1.2.840.10045.4.3.3, OID.1.2.840.10045.4.3.3) SHA512withECDSA (1.2.840.10045.4.3.4, OID.1.2.840.10045.4.4.4) NONEwithECDSA	—
	SHA1withDSA (DSA, DSS, SHA/DSA, SHA-1/DSA, SHA1/DSA, SHAwithDSA, DSAWithSHA1, 1.2.840.10040.4.3, OID.1.2.840.10040.4.3, 1.3.14.3.2.13, OID.1.3.14.3.2.13, 1.3.14.3.2.27, OID.1.3.14.3.2.27) SHA224withDSA (2.16.840.1.101.3.4.3.1, OID.2.16.840.1.101.3.4.3.1) SHA256withDSA (2.16.840.1.101.3.4.3.2, OID.2.16.840.1.101.3.4.3.2) SHA384withDSA (2.16.840.1.101.3.4.3.3, OID.2.16.840.1.101.3.4.3.3) SHA512withDSA (2.16.840.1.101.3.4.3.4, OID.2.16.840.1.101.3.4.3.4) NONEwithDSA (RawDSA)	—
Mac	HmacSHA1 (1.2.840.113549.2.7, OID.1.2.840.113549.2.7) HmacSHA224 (1.2.840.113549.2.8, OID.1.2.840.113549.2.8) HmacSHA256 (1.2.840.113549.2.9, OID.1.2.840.113549.2.9) HmacSHA384 (1.2.840.113549.2.10, OID.1.2.840.113549.2.10) HmacSHA512 (1.2.840.113549.2.11, OID.1.2.840.113549.2.11)	—
	HmacPBESHA1 HmacPBESHA224 HmacPBESHA256 HmacPBESHA384 HmacPBESHA512	PKCS #12 password-based encryption HMAC algorithms The key derivation function used for these algorithms is not a FIPS 140 allowed algorithm. These algorithms will be removed in a future release of Jipher. See Supported Non-FIPS 140 Allowed Algorithms.

Table 8-1 (Cont.) Algorithm Strings Supported by Jipher

Engine	Supported Algorithm Strings and Their Aliases	Notes
KeyGenerator	`HmacSHA1` (`1.2.840.113549.2.7`, `OID.1.2.840.113549.2.7`) `HmacSHA224` (`1.2.840.113549.2.8`, `OID.1.2.840.113549.2.8`) `HmacSHA256` (`1.2.840.113549.2.9`, `OID.1.2.840.113549.2.9`) `HmacSHA384` (`1.2.840.113549.2.10`, `OID.1.2.840.113549.2.10`) `HmacSHA512` (`1.2.840.113549.2.11`, `OID.1.2.840.113549.2.11`)	—

Table 8-1 (Cont.) Algorithm Strings Supported by Jipher

Engine	Supported Algorithm Strings and Their Aliases	Notes
	AES (Rijndael, 2.16.840.1.101.3.4.1, OID.2.16.840.1.101.3.4.1) AES_128/ECB/NoPadding (OID.2.16.840.1.101.3.4.1.1, 2.16.840.1.101.3.4.1.1) AES_192/ECB/NoPadding (OID.2.16.840.1.101.3.4.1.21, 2.16.840.1.101.3.4.1.21) AES_256/ECB/NoPadding (OID.2.16.840.1.101.3.4.1.41, 2.16.840.1.101.3.4.1.41) AES_128/CBC/PKCS5Padding (AES_128/CBC/PKCS7Padding, OID.2.16.840.1.101.3.4.1.2, 2.16.840.1.101.3.4.1.2) AES_192/CBC/PKCS5Padding (AES_192/CBC/PKCS7Padding, OID.2.16.840.1.101.3.4.1.22, 2.16.840.1.101.3.4.1.22) AES_256/CBC/PKCS5Padding (AES_256/CBC/PKCS7Padding, OID.2.16.840.1.101.3.4.1.42, 2.16.840.1.101.3.4.1.42) AES_128/OFB/NoPadding (OID.2.16.840.1.101.3.4.1.3, 2.16.840.1.101.3.4.1.3) AES_192/OFB/NoPadding (OID.2.16.840.1.101.3.4.1.23, 2.16.840.1.101.3.4.1.23) AES_256/OFB/NoPadding (OID.2.16.840.1.101.3.4.1.43, 2.16.840.1.101.3.4.1.43) AES_128/CFB/NoPadding (OID.2.16.840.1.101.3.4.1.4, 2.16.840.1.101.3.4.1.4) AES_192/CFB/NoPadding (OID.2.16.840.1.101.3.4.1.24, 2.16.840.1.101.3.4.1.24) AES_256/CFB/NoPadding (OID.2.16.840.1.101.3.4.1.44, 2.16.840.1.101.3.4.1.44) AES_128/GCM/NoPadding (OID.2.16.840.1.101.3.4.1.6, 2.16.840.1.101.3.4.1.6) AES_192/GCM/NoPadding (OID.2.16.840.1.101.3.4.1.26, 2.16.840.1.101.3.4.1.26)	—

Table 8-1 (Cont.) Algorithm Strings Supported by Jipher

Engine	Supported Algorithm Strings and Their Aliases	Notes
	AES_256/GCM/NoPadding (OID.2.16.840.1.101.3.4.1.46, 2.16.840.1.101.3.4.1.46) DESede (TripleDES, OID.1.2.840.113549.3.7, 1.2.840.113549.3.7)	
	SunTls12Prf SunTlsMasterSecret (SunTls12MasterSecret, SunTlsExtendedMasterSecret) SunTlsKeyMaterial (SunTls12KeyMaterial) SunTlsRsaPremasterSecret (SunTls12RsaPremasterSecret)	These non-standard KeyGenerator algorithms are needed to provide the cryptography required by the SunJSSE provider to support TLSv1.2.
AlgorithmParameters	EC (1.2.840.10045.2.1, OID.1.2.840.10045.2.1) DSA (1.2.840.10040.4.1, OID.1.2.840.10040.4.1, 1.3.14.3.2.12, OID.1.3.14.3.2.12) DH (DiffieHellman, 1.2.840.113549.1.3.1, OID.1.2.840.113549.1.3.1) RSASSA-PSS (1.2.840.113549.1.1.10, OID.1.2.840.113549.1.1.10)	—

Table 8-1 (Cont.) Algorithm Strings Supported by Jipher

Engine	Supported Algorithm Strings and Their Aliases	Notes
	AES (2.16.840.1.101.3.4.1, OID.2.16.840.1.101.3.4.1, 2.16.840.1.101.3.4.1.2, OID.2.16.840.1.101.3.4.1.2, 2.16.840.1.101.3.4.1.3, OID.2.16.840.1.101.3.4.1.3, 2.16.840.1.101.3.4.1.4, OID.2.16.840.1.101.3.4.1.4, 2.16.840.1.101.3.4.1.6, OID.2.16.840.1.101.3.4.1.6, 2.16.840.1.101.3.4.1.22, OID.2.16.840.1.101.3.4.1.22, 2.16.840.1.101.3.4.1.23, OID.2.16.840.1.101.3.4.1.23, 2.16.840.1.101.3.4.1.24, OID.2.16.840.1.101.3.4.1.24, 2.16.840.1.101.3.4.1.26, OID.2.16.840.1.101.3.4.1.26, 2.16.840.1.101.3.4.1.42, OID.2.16.840.1.101.3.4.1.42, 2.16.840.1.101.3.4.1.43, OID.2.16.840.1.101.3.4.1.43, 2.16.840.1.101.3.4.1.44, OID.2.16.840.1.101.3.4.1.44, 2.16.840.1.101.3.4.1.46 OID.2.16.840.1.101.3.4.1.46) DESede (OID.1.2.840.113549.3.7, 1.2.840.113549.3.7) GCM OAEP (1.2.840.113549.1.1.7, OID.1.2.840.113549.1.1.7)	—
	PBES2 (1.2.840.113549.1.5.13, OID.1.2.840.113549.1.5.13) PBE PBEWithSHA1AndDESede (OID.1.2.840.113549.1.12.1.3, 1.2.840.113549.1.12.1.3)	The key derivation function used for the PBEWithSHA1AndDESede algorithm is a not a FIPS 140 allowed algorithm. The PBEWithSHA1AndDESede algorithm will be removed in a future release of Jipher. See Supported Non-FIPS 140 Allowed Algorithms.
	PBEWithHmacSHA1AndAES_128 PBEWithHmacSHA224AndAES_128 PBEWithHmacSHA256AndAES_128 PBEWithHmacSHA384AndAES_128 PBEWithHmacSHA512AndAES_128 PBEWithHmacSHA1AndAES_256 PBEWithHmacSHA224AndAES_256 PBEWithHmacSHA256AndAES_256 PBEWithHmacSHA384AndAES_256 PBEWithHmacSHA512AndAES_256	—

Table 8-1 (Cont.) Algorithm Strings Supported by Jipher

Engine	Supported Algorithm Strings and Their Aliases	Notes
KeyPairGenerator	`RSA` (`1.2.840.113549.1.1`, `OID.1.2.840.113549.1.1`, `1.2.840.113549.1.1.1`, `OID.1.2.840.113549.1.1.1`) `RSASSA-PSS` (`PSS`, `1.2.840.113549.1.1.10`, `OID.1.2.840.113549.1.1.10`) `EC` (`EllipticCurve`, `1.2.840.10045.2.1`, `OID.1.2.840.10045.2.1`) `DSA` (`1.2.840.10040.4.1`, `OID.1.2.840.10040.4.1`, `1.3.14.3.2.12`, `OID.1.3.14.3.2.12`) `DH` (`DiffieHellman`, `1.2.840.113549.1.3.1`, `OID.1.2.840.113549.1.3.1`)	—
AlgorithmParameterGenerator	`DSA` (`1.2.840.10040.4.1`, `OID.1.2.840.10040.4.1`, `1.3.14.3.2.12`, `OID.1.3.14.3.2.12`)	—
SecretKeyFactory	`AES` `DESede` (`TripleDES`)	—

	`PBEWithHmacSHA1AndAES_128`
	`PBEWithHmacSHA224AndAES_128`
	`PBEWithHmacSHA256AndAES_128`
	`PBEWithHmacSHA384AndAES_128`
	`PBEWithHmacSHA512AndAES_128`
	`PBEWithHmacSHA1AndAES_256`
	`PBEWithHmacSHA224AndAES_256`
	`PBEWithHmacSHA256AndAES_256`
	`PBEWithHmacSHA384AndAES_256`
	`PBEWithHmacSHA512AndAES_256`
		—

	`PBKDF2WithHmacSHA1` (`PBKDF2WithSHA1`, `1.2.840.113549.1.5.12`, `OID.1.2.840.113549.1.5.12`) `PBKDF2WithHmacSHA224` (`PBKDF2WithSHA224`) `PBKDF2WithHmacSHA256` (`PBKDF2WithSHA256`) `PBKDF2WithHmacSHA384` (`PBKDF2WithSHA384`) `PBKDF2WithHmacSHA512` (`PBKDF2WithSHA512`)	—
	`PBEWithSHA1AndDESede` (`OID.1.2.840.113549.1.12.1.3`, `1.2.840.113549.1.12.1.3`)	The key derivation function used for this algorithm is a not a FIPS 140 allowed algorithm. This algorithm will be removed in a future release of Jipher. See Supported Non-FIPS 140 Allowed Algorithms.

Table 8-1 (Cont.) Algorithm Strings Supported by Jipher

Engine	Supported Algorithm Strings and Their Aliases	Notes
`KeyAgreement`	`ECDH` `DH` (`DiffieHellman`, `1.2.840.113549.1.3.1`, `OID.1.2.840.113549.1.3.1`)	—

Supported Non-FIPS 140 Allowed Algorithms

> **Note:**
>
> Support for the PKCS #12 KDF algorithm will be removed in a future Jipher release. Once Jipher no longer supports the PKCS #12 KDF algorithm, it will no longer support the following algorithms (and aliases):
>
> - `AlgorithmParameters`
>   - PBEWithSHA1AndDESede (OID.1.2.840.113549.1.12.1.3, 1.2.840.113549.1.12.1.3)
> - `Cipher`
>   - PBEWithSHA1AndDESede (OID.1.2.840.113549.1.12.1.3, 1.2.840.113549.1.12.1.3)
> - `SecretKeyFactory`
>   - PBEWithSHA1AndDESede (OID.1.2.840.113549.1.12.1.3, 1.2.840.113549.1.12.1.3)
> - `Mac`
>   - HmacPBESHA1
>   - HmacPBESHA224
>   - HmacPBESHA256
>   - HmacPBESHA384
>   - HmacPBESHA512

Jipher supports the PKCS #12 Key Derivation Function (KDF) algorithm as described in Appendix B. Deriving Keys and IVs from Passwords and Salt in *RFC 7292 - PKCS #12: Personal Information Exchange Syntax v1.1*. This algorithm is not allowed by FIPS 140. This algorithm is supported for interoperability reasons, specifically to support the following:

- Password integrity mode: Integrity is guaranteed through a Message Authentication Code (MAC) derived from a secret integrity password. The PKCS #12 KDF algorithm is used to derive a MAC key for this mode in the `Mac` algorithms HmacPBESHA1, HmacPBESHA224, HmacPBESHA256, HmacPBESHA384, and HmacPBESHA512.
- Password privacy mode: Personal information is encrypted with a symmetric key derived from a user name and a privacy password. The PKCS #12 KDF algorithm is used to derive

a decryption key for this mode in the `Cipher` algorithm PBEWithSHA1AndDESede. Note that this use of the PKCS #12 KDF algorithm is deprecated.

Keysize Restrictions

Jipher uses the following default key sizes (in bits) and enforces the following restrictions for `KeyGenerator`, `KeyPairGenerator`, and `AlgorithmParameterGenerator`.

KeyGenerator

Jipher honors the system property `jdk.security.defaultKeySize`, which enables users to configure the default key size used by `KeyGenerator`. The value of this property is a list of comma-separated entries. Each entry consists of a case-insensitive algorithm name and the corresponding default key size (in decimal) separated by a colon.

Table 8-2 KeyGenerator Algorithms and Default Key Sizes

Algorithm Name	Default Key Size	Restrictions and Comments
AES	256 if permitted by the cryptographic policy (see Import Limits on Cryptographic Algorithms), 128 otherwise.	Key size must be equal to 128, 192, or 256.
AES_128/*<mode>*/*<padding>*	128	Key size must be equal to 128.
AES_192/*<mode>*/*<padding>*	192	Key size must be equal to 192.
AES_256/*<mode>*/*<padding>*	256	Key size must be equal to 256.
DESede (Triple DES)	192	Key size must be equal to 168 or 192.
HmacSHA1	160	Key size must be at least 40 bits. Key sizes that are not a multiple of 8 are increased to the next multiple of 8.
HmacSHA224	224	Key size must be at least 40 bits. Key sizes that are not a multiple of 8 are increased to the next multiple of 8.
HmacSHA256	256	Key size must be at least 40 bits. Key sizes that are not a multiple of 8 are increased to the next multiple of 8.
HmacSHA384	384	Key size must be at least 40 bits. Key sizes that are not a multiple of 8 are increased to the next multiple of 8.
HmacSHA512	512	Key size must be at least 40 bits. Key sizes that are not a multiple of 8 are increased to the next multiple of 8.

KeyPairGenerator

Jipher honors the system property `jdk.security.defaultKeySize`, which enables users to configure the default key size used by `KeyPairGenerator`. The value of this property is a list of comma-separated entries. Each entry consists of a case-insensitive algorithm name and the corresponding default key size (in decimal) separated by a colon.

Table 8-3 KeyPairGenerator Algorithms and Default Key Sizes

Algorithm Name	Default Key Size	Restrictions and Comments
DiffieHellman	3072	Key size must be equal to 2048, 3072 or 4096. Algorithm parameter specification must specify an approved FFC Safe-prime group defined in SP 800-56A Rev. 3, "Appendix D: Approved ECC Curves and FFC Safe-prime Groups."
DSA	2048	Key size must be equal to 2048 or 3072. Algorithm parameter specification must specify one of the following (prime size, sub-prime size) domain parameter size pairings (2048, 224), (2048, 256) or (3072, 256).
EC	256	Key size must be equal to 224, 256, 384, 521. Algorithm parameter specification must specify one the four approved ECC named curves listed in Approved ECC Named Curves and SP 800-56A Rev. 3, "Appendix D: Approved ECC Curves and FFC Safe-prime Groups" defined in RFC 8422: Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier.
RSA and RSASSA-PSS	3072	Key size must be between 2,048 and 15,360 bits. The public exponent length must exceed 16 bits and cannot exceed 256 bits. If the key size exceeds 3072, then the public exponent length cannot exceed 64 bits.

Approved ECC Named Curves

Standard for Efficient Cryptography Group (SECG) Name	NIST	OID
secp224r1	P-224	1.3.132.0.33
secp256r1	P-256	1.2.840.10045.3.1.7
secp384r1	P-384	1.3.132.0.34
secp521r1	P-521	1.3.132.0.35

AlgorithmParameterGenerator

Table 8-4 AlgorithmParameterGenerator Algorithms and Default Key Sizes

Algorithm Name	Default Key Size	Restrictions and Comments
DSA	2048	Key size must be equal to 2048 or 3072. Algorithm parameter specification must specify one of the following (prime size, sub-prime size) domain parameter size pairings (2048, 224), (2048, 256), or (3072, 256).

Supported Elliptic Curve Names

Jipher supports only a fixed set of named (published) elliptic curves. These are NIST-recommended curves based on prime fields.

The following table lists the elliptic curves that are provided by Jipher.

Table 8-5 Supported Elliptic Curve Names

Elliptic Curve	Object Identifier and Aliases	Aliases
secp224r1	`1.3.132.0.33`	`P-224`, `P224`
secp256r1	`1.2.840.10045.3.1.7`	`P-256`, `P256`, `prime256v1`
secp384r1	`1.3.132.0.34`	`P-384`, `P384`
secp521r1	`1.3.132.0.35`	`P-521`, `P521`

Default Diffie-Hellman Parameters

When generating Diffie-Hellman (DH) key pairs, default DH parameters are selected based on key size. Supported key sizes are 2048, 3072, and 4096.

The default parameters are from RFC 7919: Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security.

Table 8-6 Default DH Parameters

Key Size	Default Parameter
2048	`ffdhe2048`
3072	`ffdhe3072`
4096	`ffdhe4096`

Default Digital Signature Algorithm Parameters

When generating Default Digital Signature Algorithm (DSA) key pairs, default DSA parameters are selected based on key size. Supported key sizes are 2048 and 3072.

The default parameters are verifiably generated using the FIPS 186-4 algorithm. Line breaks have been added for the values of P and G for clarity.

Table 8-7 Default DSA Parameters for the Key Size 2048

Parameter Name	Default Parameter Value
Qlen	`224`
Digest	`SHA224`

Table 8-7 (Cont.) Default DSA Parameters for the Key Size 2048

Parameter Name	Default Parameter Value
P	00F82CD0B121DF91E2F9D1A84A9A89402A40B9544184E1FDBD27B045D122D719 BF1CB7188330EA0E866D3DD2E779C81146316D7280DA9E09FFEA58F4219484B8 E7C606F8C6C15F5BD87C21730CC83484495EF991980DCE1D704C6FFB7330B691 CCEE948F39935BDF4A1E6CEDAAA6EF37C83868EA0FFA529537384E3595D14F50 FF044F9BA38CED5AB1B291D29C8DD2DA43C711E662666FE0C241835E2100C082 10FBF0E180F7941CD12C8D98BE70CD68FCC7F57D40EB447D68BA269F6A36E667 2D232B59077AD48933C95924E81C524775C7EB5E4D2996C21D7714DA89CF76C9 1C6E48E3678B80C75CE90437B3C8608886BB9595876C200CA77E554E6E0F724B 39
Q	00879D04B33B22C098583DE711AF3C6CEAB0BEBB0AAC1B5B5203154EEB
G	25895D1722207B2E06032D0269587DFDA581800D5510A5605888A7E9868BCFE6 25CFB6CFF9641AE18BDF0595CC3A7668F014D7E9A818006F7A6E63B1919A25E4 1389249F0880A968CB5E63714CA3B7CACFFB1C27BE121F7E4122FB711FCEA26F 7FE2645799A9AF6007D00F846B04242A1A9664F084BD06762C66BF2BB1E42CFD F5CAE58BD4796272150A304115ACF499FACF41F57225CA6EEDFCB909F0331B97 19E5B80F18399A677D0574BED3FDEA92BD05524B0FBCED902B73A203E26A864C 99994B19B7C93959E58D5623480349B4468B47975C0F8676F05A429DFE31D7FB 9A100F73C8B17C151391C63E814F93F7F249B7E861C7958DF56D021063DEA150
seed	767E82C5AB98153654160E06634B2B4DBE865E837C57FFD8DF03C16B
j	02
counter	112

Table 8-8 Default DSA Parameters for the Key Size 3072

Parameter Name	Default Parameter Value
Qlen	256
Digest	SHA25

Table 8-8 (Cont.) Default DSA Parameters for the Key Size 3072

Parameter Name	Default Parameter Value
P	00B4E9351B2591E98FD2AA7A1CA493E8BD6E2DD66D1BBA15871CE860EFEE68B8 AE7656E8C4D2C31F448C9F9E669D5675DC6891797F3027A1CD4F8B1E4B1B5E58 2D18CDFAED592A1C16E313CD9040A90926191D8EEA85A07A36F5D38518C90E60 1312289A755AD50ACDD302C834CC15BAA0583B8AADEE7FF1CABB28D4E7A48654 60E04A0F04DD682A35D16B56B667EF8C0C83F9E734AC1BE4041C435BA032B29A BB2B1B0D4A814AD7A8D79DA3D53CEE7CF384A7E411FC8FD88E885F7AFBA91BC0 70483971A6DBA3E66A15D789285A788AA0EFDC0CCBC75B379A5F328353D27455 41521B8E8030B0AE395B586A88AA2D591C3C303D509978E05292D0CF47298B14 C42372C394B85208B6E3D80EBAB2B53D966523F645F1F0156FEC921C049758F8 373AF0C400D761E5971C29A5720AE7419C785E7BC6BAFD00BA9A7DC75908CD42 3BBF323A092FE7BB71C77C10BA97FE1755D00D98FDFC49C86671202292D3E4AE C66E7544DAF07196C17AF4F40452B54B2E1494B9E3FD871F67B9ACCBFFA14C8C 5D
Q	0095527504A11CDD911A915EE8123BC1FE7B77EA7F9B694736907670D823AEA19D
G	00A5062CD5370449DE0274E7E9610D1F1212ADB33B78DBA9507DD62DD0BBE6CB 1C00619192388F5FD705C2C25A2D90937BB294EA3D675651A8700575058DBC00 2C62BCD9781830AAE9DCA5ACE8E563B4B4A8752B5E9EAACE284233276B5EAA25 3F2529136B9B46B2FBA732BD9EBDCAA5581910A5DA161D12EFDCDBA15B5E3A60 AE362F475C0666238B15EB1C3CFCA0D2BE85B75B47A4F790E58F977AA4FC4858 0EBBC16A0F83E9843568C886401B68025A078793F5114890D3FD0AAC5B2F5567 0013A460C052A3BBDFB72FE1E102F0B9B535DB93FAA281B9315BF5043269B1E6 C29C10A31EDBD914E3014E25347790DC71212A1313EEE4DA9AA6F933E3AEFE81 38DD0CAF20F87A75869082ED9AB8BD6E983E37F3DBFDE5E1C6DC4EC3331F649D 3E72EB5005327C7D7C604CD751D5E579A8857515F810DDD01DD8B55BC3DFB57E 79E2E2E2D1431D1DD3032598FBBBEFD7E354CFC854D3CBC37F5343B3BC6A2028 374AE3B82746A7EB7BD7D4BD933F22B85DE33B6AC3B012C3205876C28EC07215 3B
seed	DF43673D7428A318F885D40BF7B2BF6C0B977BB7E521C6CE83E347F31B28B0E5
j	02
counter	1543