Legion of the Bouncy Castle Inc.
BC-FJA (Bouncy Castle FIPS Java API)

User Guide

Version: 1.0.1 Date: 06/09/18

Legion of the Bouncy Castle Inc.
(ABN 84 166 338 567)
https://www.bouncycastle.org
Copyright and Trademark Notice

This document is licensed under a Creative Commons Attribution 3.0 Unported License (http://creativecommons.org/licenses/by/3.0/)

Sponsored By

http://www.tripwire.com/

http://www.orionhealth.com

http://www.cryptoworkshop.com

http://www.galois.com

http://www.jscape.com
Acknowledgements

Crypto Workshop would like to acknowledge that its contribution has largely been made possible through its clients purchasing Bouncy Castle support agreements.

The following people and organisations also contributed financially to this project.
Organisations: Intertrust Technologies, Lobster GmbH, PrimeKey Solutions AB, and Yozons, Inc.
Individuals: Rui Abreu, Tim Brown, Joe Brutto, Drew Carey Buglione, Andrew Carroll, Natalie Marie Ellis, Barry Faba, Khansae1992, and Brian Phelps.
To them and our anonymous donors, we are grateful. Thanks.

Validation testing was performed by InfoGard Laboratories. For more information on validation or revalidations of the software contact:

Marc Ireland
FIPS Program Manager, CISSP
InfoGard Laboratories
709 Fiero Lane, Suite 25
San Luis Obispo, CA 93401
United States of America

For further information about this distribution, or to help support this work further, please contact us at office@bouncycastle.org.
# Table of Contents

1 Introduction.............................................................................................................................7
  1.1 About the FIPS Validation.....................................................................................................7
  1.2 Use with the BC PKIX, OpenPGP (PG), and SMIME APIs....................................................7
  1.3 Commercial Support..............................................................................................................7
  1.4 Using the Module in an Application for FIPS compliance....................................................7
  1.5 If you need Changes or Enhancements................................................................................8
2 Installation...................................................................................................................................9
  2.1 Provider installation into the JRE..........................................................................................9
  2.2 Provider installation as a provider for the JSSE.................................................................10
  2.3 Provider configuration..........................................................................................................10
3 Cipher Algorithms (Symmetric)..............................................................................................12
  3.1 Available in Approved Mode Operation................................................................................12
  3.2 Available in General Operation............................................................................................12
  3.3 Paddings Available...............................................................................................................13
      3.3.1 AES Encryption using CBC and PKCS5/7Padding.........................................................14
      3.3.2 JCE AES Encryption using CBC and PKCS5/7Padding..................................................14
      3.3.3 JCE AES Encryption using GCM and an AEADParameterSpec......................................15
      3.3.4 JCE AES Encryption using CTR a short IV.....................................................................15
4 Cipher Algorithms (Public Key)..............................................................................................17
  4.1 Available in General Operation............................................................................................17
  4.2 Paddings Available...............................................................................................................17
  4.3 Examples................................................................................................................................17
      4.3.1 JCE RSA with PKCS1 Padding.......................................................................................17
      4.3.1 JCE ElGamal with OAEP SHA1 Padding..........................................................................17
5 Key Agreement Algorithms......................................................................................................19
  5.1 Available in Approved Mode Operation................................................................................19
  5.2 Available in General Operation............................................................................................21
      5.3 Examples..........................................................................................................................21
          5.3.1 Basic Agreement.......................................................................................................21
          5.3.2 Basic Agreement with Cofactor..................................................................................22
          5.3.3 Basic Agreement with PRF........................................................................................22
          5.3.4 JCE Basic Agreement................................................................................................22
          5.3.5 JCE One-pass MQV..................................................................................................23
          5.3.6 JCE One-pass MQV with key confirmation.................................................................23
6 Key Derivation Functions..........................................................................................................25
  6.1 Available in Approved Mode Operation................................................................................25
  6.2 Available in General Mode Operation..................................................................................25
      6.3 Examples..........................................................................................................................25
          6.3.1 Feedback Mode..........................................................................................................25
          6.3.2 X9.63 KDF..................................................................................................................26
7 Key Stores..................................................................................................................................27
  7.1 Examples..............................................................................................................................27
      7.1.1 BCFKS key store............................................................................................................27
8 Key Transport Algorithms.........................................................................................................28
  8.1 Examples..............................................................................................................................28
      8.1.1 JCE KTS-KEM-KWS......................................................................................................28
      8.1.2 JCE KTS-KEM-KWS in approved mode.........................................................................28
9 Key Wrapping Algorithms........................................................................................................30
  9.1 Available in Approved Mode Operation................................................................................30
9.2 Available in General Operation
9.3 Examples
  9.3.1 Key Wrapping using RSA
  9.3.2 Key Wrapping using AES
  9.3.3 JCE Key Wrapping using ElGamal
  9.3.4 JCE Key Wrapping using Camellia with padding
10 Mac Algorithms
  10.1 Available in Approved Mode Operation
  10.2 Available in General Operation
  10.3 Examples
    10.3.1 AES using CMAC – 64 bit
    10.3.2 JCE HMAC-SHA256
11 Message Digest Algorithms
  11.1 Available in Approved Mode Operation
  11.2 Available in General Operation
  11.3 Examples
    11.3.1 Use of SHA-256
    11.3.2 Use of SHAKE128
    11.3.3 JCA use of SHA3-224
12 Password Based Key Derivation Functions
  12.1 Available in Approved Mode Operation
  12.2 Available in General Operation
  12.3 Examples
    12.3.1 PBKDF2
    12.3.2 JCE PBKDF2
13 Random Number Generators
  13.1 Available in Approved Mode Operation
  13.2 Available in General Operation
  13.3 Examples
    13.3.1 Creation for SHA512 DRBG
    13.3.2 Creation for AES X9.31
14 Signature Algorithms
  14.1 Available in Approved Mode Operation
    14.1.1 DSA
    14.1.2 EC
    14.1.3 RSA
  14.2 Available in General Operation
    14.2.1 DSA
    14.2.2 DSTU4145
    14.2.3 ECDSA
    14.2.4 GOST3410-1994
    14.2.5 GOST3410-2001
    14.2.6 RSA
  14.3 Examples
    14.3.1 RSA with SHA-1
    14.3.2 JCA ECDSA with SHA-256
Appendix A – System Properties
Appendix B – Policy Permissions
  B.1 Java Permissions
  B.2 Optional Permissions
    B.2.1 Configuration of Approved/Unapproved Modes
    B.2.2 Use of CryptoServicesRegistrar.setApprovedMode(true)
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.2.3 Key Export and Translation</td>
<td>54</td>
</tr>
<tr>
<td>B.2.4 SSL Support</td>
<td>54</td>
</tr>
<tr>
<td>B.2.5 Setting of Default SecureRandom</td>
<td>55</td>
</tr>
<tr>
<td>B.2.6 Setting CryptoServicesRegistrar Properties</td>
<td>55</td>
</tr>
<tr>
<td>Appendix C – Built in Curves</td>
<td>56</td>
</tr>
<tr>
<td>Appendix D – Use with OSGI</td>
<td>59</td>
</tr>
<tr>
<td>Appendix E – Public/Private Key Conversion</td>
<td>60</td>
</tr>
<tr>
<td>Appendix F – The BCFKS file Format</td>
<td>61</td>
</tr>
<tr>
<td>Appendix G – Troubleshooting</td>
<td>63</td>
</tr>
<tr>
<td>Appendix H – Disclosures</td>
<td>64</td>
</tr>
<tr>
<td>Appendix I – References</td>
<td>65</td>
</tr>
</tbody>
</table>
1 Introduction

This document is a guide to the use of the Legion of the Bouncy Castle FIPS Java module. It is a companion document to the “Legion of the Bouncy Castle Security Policy”. As an addendum to the security policy this document is meant to provide more detail on the module. It should be noted that the security policy is the authoritative source, and in event of a conflict between this document and the security policy, the version in the security policy should be accepted. Only the security policy has been reviewed and approved by the Cryptographic Module Validation Program (CMVP), the joint US – Canadian program for the validation of cryptographic products that overviews the FIPS process.

This document assumes you are familiar with Java and at least have some familiarity with the Java Cryptography Architecture (the JCA) and the Java Cryptography Extension (the JCE).

1.1 About the FIPS Validation

The BC FIPS jar has been designed and implemented to meet FIPS 140-2, Level 1 requirements. As detailed in the security policy, providing you use the BC FIPS jar as is, the validation of the module applies to its use on the Java SE Runtime Environment 7 and the Java SE Runtime Environment 8. This is possible because the FIPS Implementation Guidance (the IG) allows affirmation of compliance providing a software module is run on the same single user environment as it was originally tested on, even if what underlies the single user environment is different (see section G.5). In our case the Java Virtual Machine represents the single user environment.

1.2 Use with the BC PKIX, OpenPGP (PG), and SMIME APIs

Separate JARs are provided for the additional Bouncy Castle APIs, these are the same as the regular ones, however the lightweight BC support classes have been removed and some internal version numbers have been changed in order to allow the BC FIPS jar and the jars for the additional APIs to work together correctly in OSGI compliant containers. The BC FIPS jar is compatible with the extra Bouncy Castle APIs from BC 1.53.

1.3 Commercial Support

Commercial support contracts for this software are available from Crypto Workshop and also help support this project. The commercial support program also entitles you to early access to the next version of the FIPS module, any updates as soon as they are made, and the opportunity to request changes and enhancements. For further information about commercial support please email us info@cryptoworkshop.com.

1.4 Using the Module in an Application for FIPS compliance

In order to meet FIPS requirements, if you develop an application with this module you need to make sure you follow the security policy provided with the module. The security policy represents the official reviewed document of how to make use of this module properly – it is the final word on the topic. If you need help following the security policy, or wish to confirm you are doing so correctly, we are happy to certify this via review of your application. If you are a support contract holder this is regarded as a consulting assignment, but you can use your consulting time towards it, and benefit from our discounted rate if any additional time is required.
1.5 If you need Changes or Enhancements

If the certification does not cover your Java runtime environment, or you need a different, possibly smaller, set of algorithms, we can help you. The module has been designed to subset and extend easily, and we are the best placed to do it – after all we designed it, and we would hope that the presence of this document shows we have definitely got the experience and we deliver.

We do not object to being involved in private validations either. We accept that there are sometimes situations where a private validation may make good sense for an organisation, either for competitive advantage, marketing, or risk management reasons. Often enhancements and sub-setting can be done using fixed rate contracts to help manage costs, and, again, the consulting time available in support contracts may be used towards this as well. So if you need further work around the module, please contact us and we will let you know what can be done.
2 Installation

The bc-fips jar can be installed in either jre/lib/ext for your JRE/JDK, or in many cases, on the general class path defined for the application you are running. In the event you do install on the bc-fips jar on the general class path be aware that sometimes the system class loader may make it impossible to use the classes in the bc-fips jar without causing an exception depending on how elements of the JCA/JCE are loaded elsewhere in the application.

Note: in any of the cases below it is likely you will need to install the unrestricted policy files for the JCE in order to make full use of the algorithm set in the bc-fips jar.

2.1 Provider installation into the JRE

Once the bcfips jar is installed, the provider class BouncyCastleFipsProvider may need to be installed if it is required in the application globally.

Installation of the provider can be done statically in the JVM by adding it to the provider definition to the java.security file in in the jre/lib/security directory for your JRE/JDK.

The provider can also be added during execution. If you wish to add the provider to the JVM globally during execution you can add the following imports to your code:

```java
import java.security.Security
import org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider
```

Then insert the line

```
Security.addProvider(new BouncyCastleFipsProvider())
```

The provider can then be used by referencing the name “BCFIPS”, for example:

```
Cipher c = Cipher.getInstance(“AES/CBC/PKCS5Padding”, “BCFIPS”);
```

Alternately if you do not wish to install the provider globally, but use it locally instead, it is possible to pass the provider to the getInstance() method on the JCA/JCE class you are creating an instance of. For example:

```
Cipher c = Cipher.getInstance(“AES/CBC/PKCS5Padding”,
                           new BouncyCastleFipsProvider());
```

Note: if the provider object is created in FIPS approved mode then only FIPS approved mode algorithms will be available.
2.2 Provider installation as a provider for the JSSE

The provider can also be used to run the JSSE in FIPS mode if the host JVM supports it (JDK 1.6 or later). In this case the provider name needs to be passed to the constructor of the JSSE provider either via the java.security file for the JVM:

```
security.provider.4=com.sun.net.ssl.internal.ssl.Provider BCFIPS
```

or using the JSSE provider constructor:

```
new com.sun.net.ssl.internal.ssl.Provider("BCFIPS")
```

Further details on using the JSSE in FIPS mode can be found at:


The JSSE makes use of Sun internal classes for passing parameters used by the TLS KDFs. In the event you are using the provider with the JSSE and there is a security manager present you will also need to grant:

```
permission java.lang.RuntimePermission
   "accessClassInPackage.sun.security.internal.spec"
```

to allow the BC FIPS provider to handle the internal classes the JSSE presents.

It should also be noted here that RSA PKCS#1.5 key wrap, NONEwithDSA, NONEwithECDSA, and NONEwithRSA require 2 additional policy settings if the BCFIPS provider is run with a SecurityManager present and in “FIPS only” mode – as a rule these algorithms are not FIPS approved, except where used for TLS and the policy settings reflect this. In the most general case this will need:

```
permission org.bouncycastle.crypto.CryptoServicesPermission "tlsAlgorithmsEnabled";
```
to be in the policy file associated with the deployment.

As the JSSE requires the presence of the SUN provider in FIPS mode, the minimal static provider configuration to support the JSSE is:

```
security.provider.1=org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider
security.provider.2=com.sun.net.ssl.internal.ssl.Provider BCFIPS
security.provider.3=sun.security.provider.Sun
```

When using the JSSE in FIPS mode you will probably also find it requires the server and client private keys to be coming from a key store supported by the BCFIPS provider. Use the BCFKS for the key store format.

2.3 Provider configuration

The provider can also take a string as a constructor argument, either via the java.security file definition or when the constructor is called at runtime.

At the moment the configuration string is limited to setting the DRBG. The configuration string must always start with “C:” and finish with “ENABLE{ALL}”.

```
new BouncyCastleFipsProvider("C:DEFRND[SHA256];ENABLE{ALL};");
```
The DRBG type string is based on the available DRBGs from SP 800-90A. The possible values are: “SHA1”, “SHA224”, “SHA256”, “SHA384”, “SHA512”, “SHA512(224)”, “SHA512(256)”, “HMACSHA1”, “HMACSHA224”, “HMACSHA256”, “HMACSHA384”, “HMACSHA512”, “CTRAES128”, “CTRAES192”, “CTRAES256”, and “CTRDESEDE”.

**Note:** In order to make the default DRBG suitable for key generation, the default DRBG is configured to be prediction resistant and this can strain the JVMs entropy source especially if hardware RNG is not available.

In situations where the amount of entropy is constrained the default DRBG for the provider can be configured to use an entropy pool based on a SHA-512 SP 800-90A DRBG. To configure this use:

```
C:HYBRID;ENABLE{All} ;
```

or include the string "HYBRID;" in the previous command string setting the DRBG. After initial seeding the entropy pool will start a reseeding thread which it will begin polling once 20 samples have been taken since the last seeding and will do a reseed as soon as new entropy bytes are returned.
3 Cipher Algorithms (Symmetric)

Cipher availability and the modes of the ciphers available depends on whether the module is running a approved-only mode or not. Not all widely used cipher modes are recognised by FIPS so in the FIPS ciphers appear in both the approved mode and general mode table, as some extra modes of operation become available for FIPS ciphers if the module in not running in approved only mode.

### 3.1 Available in Approved Mode Operation

Only the FIPS recognised ciphers and modes are available in approved mode of operation.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Low-level Class</th>
<th>Modes</th>
<th>JCE Cipher Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>FipsAES</td>
<td>ECB, CBC, CFB8, CFB128, OFB, CTR, CCM, GCM</td>
<td>AES</td>
</tr>
<tr>
<td>TripleDES</td>
<td>FipsTripleDES</td>
<td>ECB, CBC, CFB8, CFB64, OFB, CTR</td>
<td>DESede</td>
</tr>
</tbody>
</table>

### 3.2 Available in General Operation

In general operation there are a wider range of ciphers available, in addition support is also provided for the FIPS recognised ciphers to use non-FIPS modes such as OpenPGPCFB.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Low-level Class</th>
<th>Modes</th>
<th>JCE Cipher Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>AES</td>
<td>EAX, OCB, OpenPGPCFB</td>
<td>AES</td>
</tr>
<tr>
<td>ARC4</td>
<td>ARC4</td>
<td>N/A</td>
<td>ARC4, RC4</td>
</tr>
<tr>
<td>Blowfish</td>
<td>Blowfish</td>
<td>CBC, CFB8, CFB64, CTR, EAX, ECB, OFB, OpenPGPCFB</td>
<td>Blowfish</td>
</tr>
<tr>
<td>Camellia</td>
<td>Camellia</td>
<td>CBC, CCM, CFB8, CFB128, CTR, EAX, ECB, GCM, OCB, OFB</td>
<td>Camellia</td>
</tr>
<tr>
<td>CAST5</td>
<td>CAST5</td>
<td>CBC, CFB8, CFB64, CTR, EAX, ECB, OFB, OpenPGPCFB</td>
<td>CAST5</td>
</tr>
<tr>
<td>DES</td>
<td>DES</td>
<td>CBC, CFB8, CFB64, CTR, EAX, ECB, OFB, OpenPGPCFB</td>
<td>DES</td>
</tr>
<tr>
<td>GOST28147</td>
<td>GOST28147</td>
<td>CBC, CFB8, CFB64, CTR, EAX, ECB, GCFB, GOFB, OFB</td>
<td>GOST28147</td>
</tr>
<tr>
<td>IDEA</td>
<td>IDEA</td>
<td>CBC, CFB8, CFB64, CTR, EAX, ECB, OFB, OpenPGPCFB</td>
<td>IDEA</td>
</tr>
<tr>
<td>RC2</td>
<td>RC2</td>
<td>CBC, CFB8, CFB64, CTR, EAX, ECB, OFB</td>
<td>RC2</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Low-level Class</td>
<td>Modes</td>
<td>JCE Cipher Name</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------</td>
<td>--------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>SEED</td>
<td>SEED</td>
<td>CBC, CCM, CFB8, CFB128, CTR, EAX, ECB, GCM, OCB, OFB</td>
<td>SEED</td>
</tr>
<tr>
<td>Serpent</td>
<td>Serpent</td>
<td>CBC, CCM, CFB8, CFB128, CTR, EAX, ECB, GCM, OCB, OFB</td>
<td>Serpent</td>
</tr>
<tr>
<td>SHACAL-2</td>
<td>SHACAL2</td>
<td>CBC, CFB8, CFB256, CTR, EAX, ECB, OFB</td>
<td>SHACAL-2, SHACAL2</td>
</tr>
<tr>
<td>Triple-DES</td>
<td>TripleDES</td>
<td>EAX, OpenPGPCFB</td>
<td>DESede</td>
</tr>
<tr>
<td>Twofish</td>
<td>Twofish</td>
<td>CBC, CCM, CFB8, CFB128, CTR, EAX, ECB, GCM, OCB, OFB</td>
<td>Twofish</td>
</tr>
</tbody>
</table>

### 3.3 Paddings Available

FIPS is largely ambivalent towards padding mechanisms, so all the padding modes are available in both approved-mode and general operation.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Padding Type</th>
<th>JCE Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECB</td>
<td>NONE, PKCS7 (PKCS5), ISO10126-2, X9.23, ISO7816-4, TBC</td>
<td>NoPadding, PKCS7Padding (PKCS5Padding), ISO10126-2Padding, X9.23Padding, ISO7816-4Padding, TBCPadding</td>
</tr>
<tr>
<td>CBC</td>
<td>NONE, PKCS7 (PKCS5), ISO10126-2, X9.23, ISO7816-4, TBC, CS1, CS2, CS3</td>
<td>NoPadding, PKCS7Padding (PKCS5Padding), ISO10126-2Padding, ISO7816-4Padding, X9.23Padding, TBCPadding, CS1Padding, CS2Padding, CS3Padding</td>
</tr>
</tbody>
</table>

In addition to the use of PKCS5Padding for PKCS7Padding, ISO9797-1Padding can also be used as an alias for ISO7816-4Padding.

While the padding types named after standards are self explanatory, TBC is “trailing bit complement” as defined in Appendix A, NIST SP 800-38A, CS1, CS2, and CS3 are all cipher text stealing modes as defined in the Addendum to NIST SP 800-38A. CBC with CS3 is also equivalent to CTS mode in RFC 2040.

### 3.3 Examples

The low-level examples make use of the following utility methods:

```java
static byte[] encryptBytes(
    FipsOutputEncryptor outputEncryptor, byte[] plainText) throws IOException {
    ByteArrayOutputStream bOut = new ByteArrayOutputStream();
    CipherOutputStream encOut = outputEncryptor.getEncryptingStream(bOut);

    encOut.update(plainText);
    encOut.close();
```
return bOut.toByteArray();
}

and:

static byte[] decryptBytes(FipsInputDecryptor inputDecryptor,
   byte[] cipherText) throws IOException {
    ByteArrayOutputStream bOut = new ByteArrayOutputStream();
    InputStream encIn = inputDecryptor.getDecryptingStream(  
        new ByteArrayInputStream(cipherText));
    int ch;
    while ((ch = encIn.read()) >= 0)
    {
      bOut.write(ch);
    }
    return bOut.toByteArray();
}

3.3.1 AES Encryption using CBC and PKCS5/7Padding

// ensure a FIPS DRBG in use.
CryptoServicesRegistrar.setSecureRandom(  
    FipsDRBG.SHA512_HMAC.fromEntropySource(  
        new BasicEntropySourceProvider(new SecureRandom(), true))
    .build(null, true));

byte[] iv = new byte[16];
CryptoServicesRegistrar.getSecureRandom().nextBytes(iv);

FipsSymmetricKeyGenerator<SymmetricSecretKey> keyGen =  
    new FipsAES.KeyGenerator(128,  
        CryptoServicesRegistrar.getSecureRandom());

SymmetricSecretKey key = keyGen.generateKey();

FipsSymmetricOperatorFactory<FipsAES.Parameters> fipsSymmetricFactory =  
    new FipsAES.OperatorFactory();

FipsOutputEncryptor<FipsAES.Parameters> outputEncryptor =  
    fipsSymmetricFactory.createOutputEncryptor(key,  
        FipsAES.CBCwithPKCS7.withIV(iv));

byte[] output = encryptBytes(outputEncryptor, new byte[16]);

FipsInputDecryptor<FipsAES.Parameters> inputDecryptor =  
    fipsSymmetricFactory.createInputDecryptor(key,  
        FipsAES.CBCwithPKCS7.withIV(iv));

byte[] plain = decryptBytes(inputDecryptor, output);

3.3.2 JCE AES Encryption using CBC and PKCS5/7Padding

A note on this example: it will use the default provider DRBG if nothing has already been configured.
KeyGenerator keyGen = KeyGenerator.getInstance("AES", "BCFIPS");
keyGen.init(256);
SecretKey aesKey = keyGen.generateKey();

byte[] data = Hex.decode("000102030405060708090A0B0C0D0E0F");
Cipher enc = Cipher.getInstance("AES/CBC/PKCS5Padding", "BCFIPS");
enc.init(Cipher.ENCRYPT_MODE, aesKey);

byte[] encrypted = enc.doFinal(data);
byte[] iv = enc.getIV();
Cipher dec = Cipher.getInstance("AES/CBC/PKCS5Padding", "BCFIPS");
dec.init(Cipher.DECRYPT_MODE, aesKey, new IvParameterSpec(iv));

byte[] plain = dec.doFinal(encrypted);

3.3.3 JCE AES Encryption using GCM and an AEADParameterSpec

If you have to use JDK 1.5 or 1.6 there is no API in the Cipher class for introducing associated data. The AEADParameterSpec is provided to allow associated data to be prepended to the cipher text. In JDK 1.7, or later, Cipher.updateAAD() and the GCMParameterSpec can be used instead.

KeyGenerator keyGen = KeyGenerator.getInstance("AES", "BCFIPS");
keyGen.init(256);
SecretKey aesKey = keyGen.generateKey();

byte[] data = Hex.decode("000102030405060708090A0B0C0D0E0F");
byte[] assocData = Hex.decode("10111213141516171819");
byte[] nonce = Hex.decode("202122232425262728292a2b2c");
Cipher enc = Cipher.getInstance("AES/GCM/NoPadding", "BCFIPS");
enc.init(Cipher.ENCRYPT_MODE, aesKey,
        new AEADParameterSpec(nonce, 96, assocData));

byte[] encrypted = enc.doFinal(data);
Cipher dec = Cipher.getInstance("AES/GCM/NoPadding", "BCFIPS");
dec.init(Cipher.DECRYPT_MODE, aesKey,
        new AEADParameterSpec(nonce, 96, assocData));

byte[] plain = dec.doFinal(encrypted);

3.3.4 JCE AES Encryption using CTR a short IV

CTR mode can be used either with an internal counter or a provided one. An internal counter that limits the number of blocks that can be processed can be configured by providing the cipher with an IV which is less than the algorithm's block size. This will configure a limiting counter that is sizeof(cipher block) – sizeof(IV) bytes long. If the IV provided to CTR mode is the length of the block size of the algorithm the module will assume you are using an external (to the module) counter and simply add 1 to the IV as each block goes through.
If you are using a limiting counter, over, or under, flowing the counter will result in an
IllegalStateException. The maximum size for a limiting counter is 8 bytes. In the example below an
limiting counter of 4 bytes will be used as the IV is 12 bytes long, and the block size of AES is 16
bytes.

```java
KeyGenerator keyGen = KeyGenerator.getInstance("AES", "BCFIPS");
keyGen.init(256);
SecretKey aesKey = keyGen.generateKey();

byte[] data = Hex.decode("000102030405060708090A0B0C0D0E0F");
byte[] iv = Hex.decode("000102030405060708090a0b");

Cipher enc = Cipher.getInstance("AES/CTR/NoPadding", "BCFIPS");
enc.init(Cipher.ENCRYPT_MODE, aesKey, new IvParameterSpec(iv));

byte[] encrypted = enc.doFinal(data);

Cipher dec = Cipher.getInstance("AES/GCM/NoPadding", "BCFIPS");
dec.init(Cipher.DECRYPT_MODE, aesKey, new IvParameterSpec(iv));

byte[] plain = dec.doFinal(encrypted);
```
4 Cipher Algorithms (Public Key)

There are no direct public/private key ciphers available in approved mode. Available ciphers are restricted to use for key wrapping and key transport, see section 7 and section 8 for details.

4.1 Available in General Operation

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Low-level Class</th>
<th>JCE Name</th>
<th>Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELGAMAL</td>
<td>ElGamal</td>
<td>ElGamal</td>
<td>NONE, ECB</td>
</tr>
<tr>
<td>RSA</td>
<td>RSA</td>
<td>RSA</td>
<td>NONE, ECB</td>
</tr>
</tbody>
</table>

4.2 Paddings Available

Public key algorithms support the following padding mechanisms:

- NoPadding
- OAEPwithSHA-1andMGF1Padding – aliases: OAEPwithSHA1andMGF1Padding, OAEPPadding.
- OAEPwithSHA-224andMGF1Padding – aliases: OAEPwithSHA224andMGF1Padding
- OAEPwithSHA-256andMGF1Padding – aliases: OAEPwithSHA256andMGF1Padding
- OAEPwithSHA-384andMGF1Padding – aliases: OAEPwithSHA384andMGF1Padding
- OAEPwithSHA-512andMGF1Padding – aliases: OAEPwithSHA512andMGF1Padding
- PKCS1Padding

4.3 Examples

4.3.1 JCE RSA with PKCS1 Padding

```java
public byte[] pkcs1Encrypt(RSAPublicKey pubKey, byte[] data) {
    Cipher c = Cipher.getInstance("RSA/NONE/PKCS1Padding", "BCFIPS");
    c.init(Cipher.ENCRYPT_MODE, pubKey);
    return c.doFinal(data);
}
```

4.3.1 JCE ElGamal with OAEP SHA1 Padding

```java
public byte[] oaepEncrypt(DHPublicKey pubKey, byte[] data) {
    Cipher c = Cipher.getInstance("ELGAMAL/NONE/OAEPwithSHA1andMGF1Padding", "BCFIPS");
```
c.init(Cipher.ENCRYPT_MODE, pubKey);

return c.doFinal(data);
}
5 Key Agreement Algorithms

Key agreement algorithms are available based around both elliptic curve and the regular prime fields. The basic primitives are provided in the low-level classes together with support for the use of a digest or a PRF as detailed in NIST SP 800-56C.

At the JCE level the provider supports key confirmation as well. The use of key confirmation is signalled by an algorithm string containing a “/” being passed to KeyAgreement.generateSecret() which returns a new type of SecretKey - an AgreedKeyWithMacKey (see example 5.3.6 for details). The AgreedKeyWithMacKey has an additional method on it getMacKey() which returns a key that can be used for the MAC calculation in the confirmation step. In line with NIST recommendations the MAC key can be pro-actively zeroized.

5.1 Available in Approved Mode Operation

<table>
<thead>
<tr>
<th>Key Agreement Method</th>
<th>KDF</th>
<th>Low-level Class</th>
<th>JCE Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DH</td>
<td></td>
<td>FipsDH</td>
<td>DH</td>
</tr>
<tr>
<td>DHwithSHA1KDF</td>
<td>X9.63(SHA-1)</td>
<td></td>
<td>DHwithSHA1KDF</td>
</tr>
<tr>
<td>DHwithSHA224KDF</td>
<td>X9.63(SHA-224)</td>
<td></td>
<td>DHwithSHA224KDF</td>
</tr>
<tr>
<td>DHwithSHA256KDF</td>
<td>X9.63(SHA-256)</td>
<td></td>
<td>DHwithSHA256KDF</td>
</tr>
<tr>
<td>DHwithSHA384KDF</td>
<td>X9.63(SHA-384)</td>
<td></td>
<td>DHwithSHA384KDF</td>
</tr>
<tr>
<td>DHwithSHA512KDF</td>
<td>X9.63(SHA-512)</td>
<td></td>
<td>DHwithSHA512KDF</td>
</tr>
<tr>
<td>DHwithSHA512(224)KDF</td>
<td>X9.63(SHA-512(224))</td>
<td></td>
<td>DHwithSHA512(224)KDF</td>
</tr>
<tr>
<td>DHwithSHA512(256)KDF</td>
<td>X9.63(SHA-512(256))</td>
<td></td>
<td>DHwithSHA512(256)KDF</td>
</tr>
<tr>
<td>DHwithSHA1CKDF</td>
<td>Concatenation (SHA-1)</td>
<td></td>
<td>DHwithSHA1CKDF</td>
</tr>
<tr>
<td>DHwithSHA224CKDF</td>
<td>Concatenation (SHA-224)</td>
<td></td>
<td>DHwithSHA224CKDF</td>
</tr>
<tr>
<td>DHwithSHA256CKDF</td>
<td>Concatenation (SHA-256)</td>
<td></td>
<td>DHwithSHA256CKDF</td>
</tr>
<tr>
<td>DHwithSHA384CKDF</td>
<td>Concatenation (SHA-384)</td>
<td></td>
<td>DHwithSHA384CKDF</td>
</tr>
<tr>
<td>DHwithSHA512CKDF</td>
<td>Concatenation (SHA-512)</td>
<td></td>
<td>DHwithSHA512CKDF</td>
</tr>
<tr>
<td>DHwithSHA512(224)CKDF</td>
<td>Concatenation (SHA-512(224))</td>
<td></td>
<td>DHwithSHA512(224)CKDF</td>
</tr>
<tr>
<td>DHwithSHA512(256)CKDF</td>
<td>Concatenation (SHA-512(256))</td>
<td></td>
<td>DHwithSHA512(256)CKDF</td>
</tr>
<tr>
<td>MQV</td>
<td></td>
<td>FipsDH</td>
<td>MQVwithSHA1KDF</td>
</tr>
<tr>
<td>MQVwithSHA1KDF</td>
<td>X9.63(SHA-1)</td>
<td></td>
<td>MQVwithSHA1KDF</td>
</tr>
<tr>
<td>MQVwithSHA224KDF</td>
<td>X9.63(SHA-224)</td>
<td></td>
<td>MQVwithSHA224KDF</td>
</tr>
<tr>
<td>Key Agreement Method</td>
<td>KDF</td>
<td>Low-level Class</td>
<td>JCE Name</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------</td>
<td>-----------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>MQVwithSHA256KDF</td>
<td>X9.63(SHA-256)</td>
<td></td>
<td>MQVwithSHA256KDF</td>
</tr>
<tr>
<td>MQVwithSHA384KDF</td>
<td>X9.63(SHA-384)</td>
<td></td>
<td>MQVwithSHA384KDF</td>
</tr>
<tr>
<td>MQVwithSHA512KDF</td>
<td>X9.63(SHA-512)</td>
<td></td>
<td>MQVwithSHA512KDF</td>
</tr>
<tr>
<td>MQVwithSHA512(224)KDF</td>
<td>X9.63(SHA-512(224))</td>
<td></td>
<td>MQVwithSHA512(224)KDF</td>
</tr>
<tr>
<td>MQVwithSHA512(256)KDF</td>
<td>X9.63(SHA-512(256))</td>
<td></td>
<td>MQVwithSHA512(256)KDF</td>
</tr>
<tr>
<td>MQVwithSHA1CKDF</td>
<td>Concatenation (SHA-1)</td>
<td></td>
<td>MQVwithSHA1CKDF</td>
</tr>
<tr>
<td>MQVwithSHA224CKDF</td>
<td>Concatenation (SHA-224)</td>
<td></td>
<td>MQVwithSHA224CKDF</td>
</tr>
<tr>
<td>MQVwithSHA256CKDF</td>
<td>Concatenation (SHA-256)</td>
<td></td>
<td>MQVwithSHA256CKDF</td>
</tr>
<tr>
<td>MQVwithSHA384CKDF</td>
<td>Concatenation (SHA-384)</td>
<td></td>
<td>MQVwithSHA384CKDF</td>
</tr>
<tr>
<td>MQVwithSHA512CKDF</td>
<td>Concatenation (SHA-512)</td>
<td></td>
<td>MQVwithSHA512CKDF</td>
</tr>
<tr>
<td>MQVwithSHA512(224)CKDF</td>
<td>Concatenation (SHA-512(224))</td>
<td></td>
<td>MQVwithSHA512(224)CKDF</td>
</tr>
<tr>
<td>MQVwithSHA512(256)CKDF</td>
<td>Concatenation (SHA-512(256))</td>
<td></td>
<td>MQVwithSHA512(256)CKDF</td>
</tr>
<tr>
<td>ECDH</td>
<td></td>
<td>FipsEC</td>
<td>ECDH</td>
</tr>
<tr>
<td>ECDHwithSHA1KDF</td>
<td>X9.63(SHA-1)</td>
<td></td>
<td>ECDHwithSHA1KDF</td>
</tr>
<tr>
<td>ECDHwithSHA224KDF</td>
<td>X9.63(SHA-224)</td>
<td></td>
<td>ECDHwithSHA224KDF</td>
</tr>
<tr>
<td>ECDHwithSHA256KDF</td>
<td>X9.63(SHA-256)</td>
<td></td>
<td>ECDHwithSHA256KDF</td>
</tr>
<tr>
<td>ECDHwithSHA384KDF</td>
<td>X9.63(SHA-384)</td>
<td></td>
<td>ECDHwithSHA384KDF</td>
</tr>
<tr>
<td>ECDHwithSHA512KDF</td>
<td>X9.63(SHA-512)</td>
<td></td>
<td>ECDHwithSHA512KDF</td>
</tr>
<tr>
<td>ECCDH</td>
<td></td>
<td>FipsEC</td>
<td>ECCDH</td>
</tr>
<tr>
<td>ECCDHWithSHA1KDF</td>
<td>X9.63(SHA-1)</td>
<td></td>
<td>ECCDHWithSHA1KDF</td>
</tr>
<tr>
<td>ECCDHWithSHA224KDF</td>
<td>X9.63(SHA-224)</td>
<td></td>
<td>ECCDHWithSHA224KDF</td>
</tr>
<tr>
<td>ECCDHWithSHA256KDF</td>
<td>X9.63(SHA-256)</td>
<td></td>
<td>ECCDHWithSHA256KDF</td>
</tr>
<tr>
<td>ECCDHWithSHA384KDF</td>
<td>X9.63(SHA-384)</td>
<td></td>
<td>ECCDHWithSHA384KDF</td>
</tr>
<tr>
<td>ECCDHWithSHA512KDF</td>
<td>X9.63(SHA-512)</td>
<td></td>
<td>ECCDHWithSHA512KDF</td>
</tr>
<tr>
<td>ECCDHWithSHA1CKDF</td>
<td>Concatenation (SHA-1)</td>
<td></td>
<td>ECCDHWithSHA1CKDF</td>
</tr>
<tr>
<td>ECCDHWithSHA224CKDF</td>
<td>Concatenation (SHA-224)</td>
<td></td>
<td>ECCDHWithSHA224CKDF</td>
</tr>
<tr>
<td>ECCDHWithSHA256CKDF</td>
<td>Concatenation (SHA-256)</td>
<td></td>
<td>ECCDHWithSHA256CKDF</td>
</tr>
<tr>
<td>Key Agreement Method</td>
<td>KDF</td>
<td>Low-level Class</td>
<td>JCE Name</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------</td>
<td>-------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>ECCDHwithSHA384CKDF</td>
<td>Concatenation (SHA-384)</td>
<td></td>
<td>ECCDHwithSHA384CKDF</td>
</tr>
<tr>
<td>ECCDHwithSHA512CKDF</td>
<td>Concatenation (SHA-512)</td>
<td></td>
<td>ECCDHwithSHA512CKDF</td>
</tr>
<tr>
<td>ECCDHwithSHA512(224)CKDF</td>
<td>Concatenation (SHA-512(224))</td>
<td></td>
<td>ECCDHwithSHA512(224)CKDF</td>
</tr>
<tr>
<td>ECCDHwithSHA512(256)CKDF</td>
<td>Concatenation (SHA-512(256))</td>
<td></td>
<td>ECCDHwithSHA512(256)CKDF</td>
</tr>
<tr>
<td>ECDH</td>
<td></td>
<td>FipsEC</td>
<td>ECDH</td>
</tr>
<tr>
<td>ECDHwithSHA1KDF</td>
<td>X9.63(SHA-1)</td>
<td></td>
<td>ECDHwithSHA1KDF</td>
</tr>
<tr>
<td>ECDHwithSHA224KDF</td>
<td>X9.63(SHA-224)</td>
<td></td>
<td>ECDHwithSHA224KDF</td>
</tr>
<tr>
<td>ECDHwithSHA256KDF</td>
<td>X9.63(SHA-256)</td>
<td></td>
<td>ECDHwithSHA256KDF</td>
</tr>
<tr>
<td>ECDHwithSHA384KDF</td>
<td>X9.63(SHA-384)</td>
<td></td>
<td>ECDHwithSHA384KDF</td>
</tr>
<tr>
<td>ECDHwithSHA512KDF</td>
<td>X9.63(SHA-512)</td>
<td></td>
<td>ECDHwithSHA512KDF</td>
</tr>
<tr>
<td>ECDHwithSHA1CKDF</td>
<td>Concatenation (SHA-1)</td>
<td></td>
<td>ECDHwithSHA1CKDF</td>
</tr>
<tr>
<td>ECDHwithSHA224CKDF</td>
<td>Concatenation (SHA-224)</td>
<td></td>
<td>ECDHwithSHA224CKDF</td>
</tr>
<tr>
<td>ECDHwithSHA256CKDF</td>
<td>Concatenation (SHA-256)</td>
<td></td>
<td>ECDHwithSHA256CKDF</td>
</tr>
<tr>
<td>ECDHwithSHA384CKDF</td>
<td>Concatenation (SHA-384)</td>
<td></td>
<td>ECDHwithSHA384CKDF</td>
</tr>
<tr>
<td>ECDHwithSHA512CKDF</td>
<td>Concatenation (SHA-512)</td>
<td></td>
<td>ECDHwithSHA512CKDF</td>
</tr>
<tr>
<td>ECDHwithSHA512(224)CKDF</td>
<td>Concatenation (SHA-512(224))</td>
<td></td>
<td>ECDHwithSHA512(224)CKDF</td>
</tr>
<tr>
<td>ECDHwithSHA512(256)CKDF</td>
<td>Concatenation (SHA-512(256))</td>
<td></td>
<td>ECDHwithSHA512(256)CKDF</td>
</tr>
</tbody>
</table>

Note: in approved mode ECDH will only work if the cofactor of the EC domain parameters being used is 1.

5.2 Available in General Operation

There are no additional key agreement algorithms offered in general operation, however ECDH can be used even with curves that do not have a cofactor of 1.

5.3 Examples

The following examples are all for elliptic curve, other than Cofactor-Diffie-Hellman (CDH), they can equally be applied to FipsDH, or the algorithm “DH” in the case of the JCE.

5.3.1 Basic Agreement

```java
public byte[] basicAgreement(```
5.3.2 Basic Agreement with Cofactor

```java
public byte[] basicAgreementWithCofactor(
    AsymmetricECPrivateKey ourKey,
    AsymmetricECPublicKey otherPartyKey)
{
    FipsEC.DHAgreementFactory agreementFactory =
        new FipsEC.DHAgreementFactory();
    FipsAgreement<FipsEC.AgreementParameters> agree =
        agreementFactory.createAgreement(ourKey, FipsEC.DH);

    return agree.calculate(otherPartyKey);
}
```

5.3.3 Basic Agreement with PRF

```java
public byte[] basicAgreementWithPRF(
    AsymmetricECPrivateKey ourKey,
    AsymmetricECPublicKey otherPartyKey,
    FipsKDF.PRF prfAlg,
    byte[] salt)
{
    FipsEC.DHAgreementFactory agreementFactory =
        new FipsEC.DHAgreementFactory();
    FipsEC.AgreementParameters params =
        new FipsEC.AgreementParameters(FipsEC.DH, prfAlg, salt);
    FipsAgreement<FipsEC.AgreementParameters> agree =
        agreementFactory.createAgreement(ourKey, params);

    return agree.calculate(otherPartyKey);
}
```

5.3.4 JCE Basic Agreement

```java
public byte[] basicAgreement(
    ECPromoteKey ourKey,
    ECPublicKey otherPartyKey)
```
KeyAgreement agree = KeyAgreement.getInstance("ECDH", "BCFIPS");
agree.init(ourKey);
agree.doPhase(otherPartyKey, true);
return agree.generateSecret();

5.3.5 JCE One-pass MQV
MQV requires the use of ephemeral keys, in the case of one-pass MQV the other party's static key is assumed to be their ephemeral key.

public byte[] onePassMQV(
    ECPublicKey ourKey,
    KeyPair ourEphemeralKeys,
    ECPublicKey otherPartyKey)
{
    KeyAgreement agree = KeyAgreement.getInstance("ECMQVwithSHA256KDF", "BCFIPS");
    agree.doPhase(otherPartyKey, true);
    return agree.generateSecret();
}

5.3.6 JCE One-pass MQV with key confirmation
The following sample returns a provider specific SecretKey type, a AgreedKeyWithMacKey, which allows a user to do key agreement with key confirmation. For all intents and purposes an AgreedKeyWithMacKey behaves like a secret key, but it also has an extra method on it for returning the MAC key associated with the agreement.

This sample also provides an example of how you can specify a key size and an algorithm with the BC FIPS provider. In the example below the the AgreedKeyWithMacKey will have a MAC key of 128 bits (algorithm name CMAC) and an encryption key of 256 bits (algorithm name AES).

public AgreedKeyWithMacKey onePassMQV(
    ECPublicKey ourKey,
    KeyPair ourEphemeralKeys,
    ECPublicKey otherPartyKey)
{
    KeyAgreement agree = KeyAgreement.getInstance("ECMQVwithSHA256KDF", "BCFIPS");
    agree.doPhase(otherPartyKey, true);
    return agree.generateSecret();
}
agree.doPhase(otherPartyKey, true);

return (AgreedKeyWithMacKey)agree.generateSecret("CMAC[128]/AES[256]");
6 Key Derivation Functions

Most of the current KDFs in SP 800-135 and SP 800-108 are supported. In addition Scrypt is available when the Java module is not running in approved-only mode.

6.1 Available in Approved Mode Operation

<table>
<thead>
<tr>
<th>Derivation Method</th>
<th>Low-level Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter Mode</td>
<td>FipsKDF</td>
</tr>
<tr>
<td>Feedback Mode</td>
<td>FipsKDF</td>
</tr>
<tr>
<td>Double Pipeline Iteration Mode</td>
<td>FipsKDF</td>
</tr>
<tr>
<td>TLS 1.0</td>
<td>FipsKDF</td>
</tr>
<tr>
<td>TLS 1.1</td>
<td>FipsKDF</td>
</tr>
<tr>
<td>TLS 1.2</td>
<td>FipsKDF</td>
</tr>
<tr>
<td>SSH</td>
<td>FipsKDF</td>
</tr>
<tr>
<td>X 9.63</td>
<td>FipsKDF</td>
</tr>
<tr>
<td>Concatenation</td>
<td>FipsKDF</td>
</tr>
<tr>
<td>IKEv2</td>
<td>FipsKDF</td>
</tr>
<tr>
<td>SRTP</td>
<td>FipsKDF</td>
</tr>
</tbody>
</table>

6.2 Available in General Mode Operation

<table>
<thead>
<tr>
<th>Derivation Method</th>
<th>Low-level Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCrypt</td>
<td>KDF</td>
</tr>
</tbody>
</table>

6.3 Examples

KDFs are currently not directly exposed in the JCE/JCA layer, although they are made use of internally by algorithms like Diffie-Hellman and also by the JSSE. They can be invoked directly using the low-level API.

6.3.1 Feedback Mode

An example of feedback mode using AES_CMAC as the PRF and an 8 bit counter. Such a KDF could be used to generate up to 4096 bytes (being 256 * 16 (the block size of AES_CMAC)). In the example below it is being used to generate 128 bits.

```java
KDFOperatorFactory kdfOperatorFactory = new FipsKDF.FeedbackModeFactory();
KDFCalculator<FipsKDF.FeedbackModeParameters> kdfCalculator =
    kdfOperatorFactory.createKDFCalculator(FipsKDF.FEEDBACK_MODE.withPRF(FipsKDF.PRF.AES_CMAC));
```
byte[] out = new byte[16];
kdfCalculator.generateBytes(out);

### 6.3.2 X9.63 KDF

This is the regular KDF used by Diffie-Hellman algorithms. In the following example it is also being used to generate 128 bits.

```java
FipsKDFOperatorFactory<FipsKDF.AgreementKDFParameters> kdfOpt = new FipsKDF.X963OperatorFactory();
KDFCalculator kdfCalculator = kdfOpt.createKDFCalculator(FipsKDF.X963.using(Z));
```

byte[] out = new byte[16];
kdfCalculator.generateBytes(out);
7 Key Stores

The BC FIPS provider supports two types of KeyStore formats.

<table>
<thead>
<tr>
<th>Key Store Type</th>
<th>Format</th>
<th>JCA Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PKCS12</td>
<td>ASN.1 BER/DER</td>
<td>PKCS12</td>
</tr>
<tr>
<td>BCFKS</td>
<td>DER</td>
<td>BCFKS</td>
</tr>
</tbody>
</table>

The PKCS12 key store is not available in approved-mode of operation due to the algorithms required for PBE key generation in the PKCS#12 standard. The PKCS12 key store supports the following variations:

- PKCS12-3DES-3DES: the default, uses Triple-DES for any encryption task.
- PKCS12-3DES-40RC2: the traditional one, uses Triple-DES for private keys and 40 bit RC2 for certificate protection.

The BCFKS key store is designed to be FIPS compliant. It is available in approved-mode operation and is also capable of storing some secret key types in addition to public/private keys and certificates. The BCFKS key store uses PBKDF2 with HMAC SHA512 for password to key conversion and AES CCM for encryption. Passwords are encoded for conversion into keys using PKCS#12 format (as in each 16 bit character is converted into 2 bytes).

7.1 Examples

7.1.1 BCFKS key store

This code fragment shows storage of a private key and its certificate chain, as well as the storage of an AES key and a Triple-DES key.

```java
PrivateKey key = ... // asymmetric private key
X509Certificate[] certs = ... // certificate chain for corresponding public key

KeyStore store1 = KeyStore.getInstance("BCFKS", "BCFIPS");
store.load(null, null);
store.setKeyEntry("privkey", key, passwd, certs);

SecretKeySpec aesKey = new SecretKeySpec(
    Hex.decode("000102030405060708090a0b0c0d0e0f"), "AES");
store.setKeyEntry("secret1", aesKey, "secretPw1".toCharArray(), null);

SecretKeySpec edeKey = new SecretKeySpec(
    Hex.decode("010203040708090a0b0c0d0e0f"), "DESede");
store.setKeyEntry("secret2", edeKey, "secretPw2".toCharArray(), null);

ByteArrayOutputStream bOut = new ByteArrayOutputStream();
store.store(bOut, passwd);
```
8 Key Transport Algorithms

The BC FIPS provider supports the following two key transport methods:

<table>
<thead>
<tr>
<th>Algorithm Name</th>
<th>Low-level Class</th>
<th>JCE Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA-KTS-KEM-KWS</td>
<td>FipsRSA</td>
<td>RSA-KTS-KEM-KWS</td>
</tr>
<tr>
<td>RSA-KTS-OAEP</td>
<td>FipsRSA</td>
<td>RSA-KTS-OAEP</td>
</tr>
</tbody>
</table>

Both algorithms are supported in the JCE using SecretKeyFactory implementations. Owing to it being used in RFC 5990, RSA-KTS-KEM-KWS is also support as a Cipher, although when used under those circumstances it is not currently possible to use it with key confirmation.

8.1 Examples

8.1.1 JCE KTS-KEM-KWS

This is an example of RSA-KTS-KEM-KWS as used in RFC 5990. In this case we are assuming the wrapped key is an AES key. The wrapping algorithm for RSA-KTS-KEM-KWS to use is specified via a KTSParameterSpec. In this example the KEM process is used to calculate an AES-256 key to wrap the AES key held in the SecretKey object.

```java
public byte[] wrapKey(RSAPublicKey publicKey, SecretKey secretKey)
    throws Exception
{
    Cipher wrapper = Cipher.getInstance("RSA-KTS-KEM-KWS", "BCFIPS");
    KTSParameterSpec ktsParameterSpec = new KTSParameterSpec.Builder(
        NISTObjectIdentifiers.id_aes256_wrap.getId(), 256).build();
    wrapper.init(Cipher.WRAP_MODE, publicKey, ktsParameterSpec);
    return wrapper.wrap(secretKey);
}

public SecretKey unwrapKey(RSAPrivateKey privateKey, byte[] wrappedKey)
    throws Exception
{
    Cipher unwrapper = Cipher.getInstance("RSA-KTS-KEM-KWS", "BCFIPS");
    KTSParameterSpec ktsParameterSpec = new KTSParameterSpec.Builder(
        NISTObjectIdentifiers.id_aes256_wrap.getId(), 256).build();
    unwrapper.init(Cipher.UNWRAP_MODE, privateKey, ktsParameterSpec);
    return (SecretKey)unwrapper.unwrap(wrappedKey, "AES", Cipher.SECRET_KEY);
}
```

8.1.2 JCE KTS-KEM-KWS in approved mode

Note that in approved mode KTS-KEM-KWS has to be provided with OtherInfo, which is a byte array providing additional data for the KDF used with KEMS that helps distinguish the exchange. Suggestions for what should go into OtherInfo are in SP 800-56A and a utility class org.bouncycastle.crypto.util.DEROtherInfo is provided for constructing OtherInfo data in line with
the common guidelines. As far as the API is concerned OtherInfo is just a byte array.

```java
public byte[] wrapKey(
    RSAPublicKey publicKey, SecretKey secretKey, byte[] otherInfo)
    throws Exception
{
    Cipher wrapper = Cipher.getInstance("RSA-KTS-KEM-KWS", "BCFIPS");
    KTSParameterSpec ktsParameterSpec = new KTSParameterSpec.Builder(
        NISTObjectIdentifiers.id_aes256_wrap.getId(), 256, otherInfo).build();
    wrapper.init(Cipher.WRAP_MODE, publicKey, ktsParameterSpec);
    return wrapper.wrap(secretKey);
}

public SecretKey unwrapKey(
    RSAPrivateKey privateKey, byte[] wrappedKey, byte[] otherInfo)
    throws Exception
{
    Cipher unwrapper = Cipher.getInstance("RSA-KTS-KEM-KWS", "BCFIPS");
    KTSParameterSpec ktsParameterSpec = new KTSParameterSpec.Builder(
        NISTObjectIdentifiers.id_aes256_wrap.getId(), 256, otherInfo).build();
    unwrapper.init(Cipher.UNWRAP_MODE, privateKey, ktsParameterSpec);
    return (SecretKey)unwrapper.unwrap(wrappedKey, "AES", Cipher.SECRET_KEY);
}
```
9 Key Wrapping Algorithms

Algorithms for key wrapping are supported for both asymmetric and symmetric keys.

9.1 Available in Approved Mode Operation

The following key wrapping techniques are available using symmetric ciphers in approved-mode.

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Algorithm</th>
<th>Low-level Class</th>
<th>JCE Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>KW</td>
<td>FipsAES</td>
<td>AESKW, AESWrap</td>
</tr>
<tr>
<td>AES</td>
<td>KWP</td>
<td>FipsAES</td>
<td>AESKWP, AESWrapPad</td>
</tr>
<tr>
<td>DESede</td>
<td>TKW</td>
<td>FipsTripleDES</td>
<td>DESedeTKW, DESedeWrap</td>
</tr>
</tbody>
</table>

The following key wrapping techniques are available using asymmetric ciphers in approved-mode.

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Algorithm</th>
<th>Low-level Class</th>
<th>JCE Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA</td>
<td>OAEP</td>
<td>FipsRSA</td>
<td>RSA/NONE/OAEPwithSHA1andMGF1Padding, RSA/NONE/OAEPwithSHA224andMGF1Padding, RSA/NONE/OAEPwithSHA256andMGF1Padding, RSA/NONE/OAEPwithSHA384andMGF1Padding, RSA/NONE/OAEPwithSHA512andMGF1Padding</td>
</tr>
</tbody>
</table>

9.2 Available in General Operation

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Algorithm</th>
<th>Low-level Class</th>
<th>JCE Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>RFC3211</td>
<td>AES</td>
<td>AESRFC3211Wrap</td>
</tr>
<tr>
<td>Camellia</td>
<td>KW</td>
<td>Camellia</td>
<td>CamelliaKW, CamelliaWrap</td>
</tr>
<tr>
<td>Camellia</td>
<td>KWP</td>
<td>Camellia</td>
<td>CamelliaKWP, CamelliaWrapPad</td>
</tr>
<tr>
<td>DESede</td>
<td>RFC3211</td>
<td>TripleDES</td>
<td>DESedeRFC3211Wrap</td>
</tr>
<tr>
<td>DESede</td>
<td>RFC3217</td>
<td>TripleDES</td>
<td>DESedeRFC3217Wrap</td>
</tr>
<tr>
<td>SEED</td>
<td>KW</td>
<td>SEED</td>
<td>SEEDKW, SEEDWrap</td>
</tr>
<tr>
<td>SEED</td>
<td>KWP</td>
<td>SEED</td>
<td>SEEDKWP, SEEDWrapPad</td>
</tr>
<tr>
<td>Serpent</td>
<td>KW</td>
<td>Serpent</td>
<td>SerpentKW, SerpentWrap</td>
</tr>
<tr>
<td>Serpent</td>
<td>KWP</td>
<td>Serpent</td>
<td>SerpentKWP, SerpentWrapPad</td>
</tr>
<tr>
<td>Twofish</td>
<td>KW</td>
<td>Twofish</td>
<td>TwofishKW, TwofishWrap</td>
</tr>
</tbody>
</table>
The following key wrapping techniques are available using asymmetric ciphers in general-mode operation.

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Algorithm</th>
<th>Low-level Class</th>
<th>JCE Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twofish</td>
<td>KWP</td>
<td>Twofish</td>
<td>TwofishKWP, TwofishWrapPad</td>
</tr>
<tr>
<td>ElGamal</td>
<td>OAEP</td>
<td>ElGamal</td>
<td>ElGamal/NONE/OAEPwithSHA1andMGF1Padding, ElGamal/NONE/OAEPwithSHA224andMGF1Padding, ElGamal/NONE/OAEPwithSHA256andMGF1Padding, ElGamal/NONE/OAEPwithSHA384andMGF1Padding, ElGamal/NONE/OAEPwithSHA512andMGF1Padding</td>
</tr>
<tr>
<td>ElGamal</td>
<td>PKCS#1</td>
<td>ElGamal</td>
<td>ElGamal/NONE/PKCS1Padding</td>
</tr>
<tr>
<td>RSA</td>
<td>PKCS#1</td>
<td>RSA</td>
<td>RSA/NONE/PKCS1Padding</td>
</tr>
</tbody>
</table>

### 9.3 Examples

#### 9.3.1 Key Wrapping using RSA

The following snippet shows an example of using RSA with OAEP to wrap a secret key in the low-level API. Note: that in both the wrapping and unwrapping process a SecureRandom is required in order to facilitate RSA blinding on the decryption.

```java
public byte[] wrapKey(AsymmetricRSAPublicKey pubKey, byte[] inputKeyBytes) {
    FipsRSA.KeyWrapOperatorFactory wrapFact = 
        new FipsRSA.KeyWrapOperatorFactory();

    FipsKeyWrapperUsingSecureRandom wrapper = 
        wrapFact.createKeyWrapper(pubKey, FipsRSA.WRAP_OAEP) .withSecureRandom(new SecureRandom());

    return wrapper.wrap(inputKeyBytes, 0, inputKeyBytes.length);
}
```

The following snippet shows an example of unwrapping a wrapped secret key that has been wrapped using RSA with OAEP in the low-level API.

```java
public byte[] unwrapKey(AsymmetricRSAPrivateKey privKey, byte[] wrappedKeyBytes) {
    FipsRSA.KeyWrapOperatorFactory wrapFact = 
        new FipsRSA.KeyWrapOperatorFactory();

    FipsKeyUnwrapperUsingSecureRandom unwrapper = 
        wrapFact.createKeyUnwrapper(privKey, FipsRSA.WRAP_OAEP) .withSecureRandom(new SecureRandom());

    return unwrapper.unwrap(wrappedKeyBytes, 0, wrappedKeyBytes.length);
}
```
9.3.2 Key Wrapping using AES

In the following example the contents of the byte array `inputKeyBytes` will be wrapped using the KW key wrapping technique from FIPS SP800-38F.

```java
byte[] inputKeyBytes = ...; // bytes making up the key to be wrapped
byte[] keyBytes = ...; // bytes making up AES key doing the wrapping
SymmetricKey aesKey = new SymmetricSecretKey(FipsAES.KW, keyBytes);
FipsAES.KeyWrapOperatorFactory factory = new FipsAES.KeyWrapOperatorFactory();
KeyWrapper wrapper = factory.createKeyWrapper(aesKey, FipsAES.KW);
byte[] wrappedBytes = wrapper.wrap(inputKeyBytes, 0, inputKeyBytes.length);
```

9.3.3 JCE Key Wrapping using ElGamal

The following snippet shows an example of using ElGamal with OAEP to wrap a secret key.

```java
public byte[] wrapKey(DHPublicKey pubKey, SecretKey keyToBeWrapped)
    throws Exception
{
    Cipher c = Cipher.getInstance(
            "ElGamal/NONE/OAEPwithSHA256andMGF1Padding", "BCFIPS");

    c.init(Cipher.WRAP_MODE, pubKey);

    return c.wrap(keyToBeWrapped);
}
```

The following snippet shows an example of unwrapping a wrapped secret key that has been wrapped using ElGamal with OAEP.

```java
public SecretKey unwrapKey(
    DHPrivateKey privKey, byte[] wrappedKey, String wrappedKeyAlgorithm)
    throws Exception
{
    Cipher c = Cipher.getInstance(
            "ElGamal/NONE/OAEPwithSHA256andMGF1Padding", "BCFIPS");

    c.init(Cipher.UNWRAP_MODE, privKey);

    return (SecretKey)c.unwrap(out, wrappedKeyAlgorithm, Cipher.SECRET_KEY);
}
```

9.3.4 JCE Key Wrapping using Camellia with padding

In the below snippets it is assumed `secKey` is a valid Camellia key. It is also worth keeping in mind that `keyToBeWrapped` can also be a PublicKey or PrivateKey.

The first snippet shows a simple function which wraps a key – `keyToBeWrapped`.

```java
public byte[] wrapKey(SecretKey secKey, SecretKey keyToBeWrapped)
    throws Exception
{
    Cipher c = Cipher.getInstance(
            "CamelliaWrapPad", "BCFIPS");
```
c.init(Cipher.WRAP_MODE, secKey);

return c.wrap(keyToBeWrapped);
}

The second snippet shows the key unwrap process. The JCE also requires an algorithm name and key type (in this case Cipher.SECRET_KEY) to unwrap a key.

public SecretKey unwrapKey(  
    SecretKey secKey, byte[] wrappedKey, String wrappedKeyAlgorithm)  
    throws Exception
{
    Cipher c = Cipher.getInstance(  
        “CamelliaWrapPad”, “BCFIPS”);

    c.init(Cipher.UNWRAP_MODE, secKey);

    return (SecretKey)c.unwrap(out, wrappedKeyAlgorithm, Cipher.SECRET_KEY);
}
10 Mac Algorithms
A broad range of MAC and HMAC algorithms are supported by the BC FIPS provider.

10.1 Available in Approved Mode Operation

<table>
<thead>
<tr>
<th>MAC Name</th>
<th>Low-level Class</th>
<th>JCE Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES CCM MAC</td>
<td>FipsAES</td>
<td>CCMMAC, AESCCMMAC, AES-CCMMAC</td>
</tr>
<tr>
<td>AES CMAC</td>
<td>FipsAES</td>
<td>CMAC, AESCMAC, AES-CMAC</td>
</tr>
<tr>
<td>AES GMAC</td>
<td>FipsAES</td>
<td>GMAC, AESGMAC, AES-GMAC</td>
</tr>
<tr>
<td>HMAC SHA-1</td>
<td>FipsSHS</td>
<td>HmacSHA1, Hmac128SHA1</td>
</tr>
<tr>
<td>HMAC SHA-224</td>
<td>FipsSHS</td>
<td>HmacSHA224, Hmac128SHA224</td>
</tr>
<tr>
<td>HMAC SHA-256</td>
<td>FipsSHS</td>
<td>HmacSHA256, Hmac128SHA256</td>
</tr>
<tr>
<td>HMAC SHA-384</td>
<td>FipsSHS</td>
<td>HmacSHA384, Hmac256SHA384</td>
</tr>
<tr>
<td>HMAC SHA-512</td>
<td>FipsSHS</td>
<td>HmacSHA512, Hmac256SHA512</td>
</tr>
<tr>
<td>HMAC SHA-512(224)</td>
<td>FipsSHS</td>
<td>HmacSHA512(224), Hmac128SHA512(224)</td>
</tr>
<tr>
<td>HMAC SHA-512(256)</td>
<td>FipsSHS</td>
<td>HmacSHA512(256), Hmac128SHA512(256)</td>
</tr>
<tr>
<td>Triple-DES CMAC</td>
<td>FipsTripleDES</td>
<td>DESedeCMAC, DESede-CMAC</td>
</tr>
</tbody>
</table>

HMAC algorithms in the form of HmacNdigestName where N is a number, produce truncated versions of the HMAC in question. The right most bits are truncated as per the NIST standards.

10.2 Available in General Operation

<table>
<thead>
<tr>
<th>MAC Name</th>
<th>Low-level Class</th>
<th>JCE Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blowfish CMAC</td>
<td>Blowfish</td>
<td>BlowfishCMAC, Blowfish-CMAC</td>
</tr>
<tr>
<td>Camellia CCM MAC</td>
<td>Camellia</td>
<td>CamelliaCCMMAC, Camellia-CCMMAC</td>
</tr>
<tr>
<td>Camellia CMAC</td>
<td>Camellia</td>
<td>CamelliaCMAC, Camellia-CMAC</td>
</tr>
<tr>
<td>Camellia GMAC</td>
<td>Camellia</td>
<td>CamelliaGMAC, Camellia-GMAC</td>
</tr>
<tr>
<td>CAST5 CMAC</td>
<td>CAST5</td>
<td>CAST5CMAC, CAST5-CMAC</td>
</tr>
<tr>
<td>DES CBC MAC</td>
<td>DES</td>
<td>DESMAC</td>
</tr>
<tr>
<td>DES CFB8 MAC</td>
<td>DES</td>
<td>DESMAC/CFB8</td>
</tr>
<tr>
<td>DES MAC 64</td>
<td>DES</td>
<td>DESMAC64</td>
</tr>
<tr>
<td>DES MAC 64 with ISO7816-4 padding</td>
<td>DES</td>
<td>DESMAC64WITHISO7816-4PADDING</td>
</tr>
<tr>
<td>DES ISO9797 MAC</td>
<td>DES</td>
<td>ISO9797ALG3MAC</td>
</tr>
</tbody>
</table>
### MAC Name | Low-level Class | JCE Name
--- | --- | ---
DES ISO9797 MAC with ISO7816-4 padding | DES | ISO9797ALG3WITHISO7816-4PADDING
GOST28147 MAC | GOST28147 | GOST28147MAC
HMAC GOST3411 | SecureHash | HmacGOST3411
HMAC RIPEMD128 | SecureHash | HmacRIPEMD128
HMAC RIPEMD160 | SecureHash | HmacRIPEMD160
HMAC RIPEMD256 | SecureHash | HmacRIPEMD256
HMAC RIPEMD320 | SecureHash | HmacRIPEMD320
HMAC Tiger | SecureHash | HmacTiger
HMAC Whirlpool | SecureHash | HmacWhirlpool
IDEA CMAC | IDEA | IDEACMAC, IDEA-CMAC
IDEA CBC MAC | IDEA | IDEAMAC
IDEA CFB8 MAC | IDEA | IDEAMAC/CFB8
SEED CCMMAC | SEED | SEEDCCMMAC, SEED-CCMMAC
SEED CMAC | SEED | SEEDCMAC, SEED-CMAC
SEED GMAC | SEED | SEEDGMAC, SEED-GMAC
Serpent CCM MAC | Serpent | SerpentCCMMAC, Serpent-CCMMAC
Serpent CMAC | Serpent | SerpentCMAC, Serpent-CMAC
Serpent GMAC | Serpent | SerpentGMAC, Serpent-GMAC
SHACAL-2 CMAC | SHACAL2 | SHACAL-2CMAC, SHACAL-2-CMAC
Triple-DES CBC MAC | TripleDES | DESedeMAC
Triple-DES CFB8 MAC | TripleDES | DESedeMAC/CFB8
Triple-DES CBC MAC 64 | TripleDES | DESedeMAC64
Triple-DES CBC MAC 64 with ISO7816-4 padding | TripleDES | DESedeMAC64withISO7816-4Padding
Twofish CCM MAC | Twofish | TwofishCCMMAC, Twofish-CCMMAC
Twofish CMAC | Twofish | TwofishCMAC, Twofish-CMAC
Twofish GMAC | Twofish | TwofishGMAC, Twofish-GMAC

### 10.3 Examples

#### 10.3.1 AES using CMAC – 64 bit.

The following snippet shows a function returning a 64 bit CMAC MAC produced from the passed in `aesKey` and `data`.

```java
public byte[] createMac(SymmetricKey aesKey, byte[] data)
    throws IOException
{
    FipsMACOperatorFactory<FipsAES.AuthParameters> fipsSymmetricFactory =
```
new FipsAES.MACOperatorFactory();
FipsOutputMACCalculator<FipsAES.AuthParameters> macCalculator =
    fipsSymmetricFactory.createOutputMACCalculator(
aesKey, FipsAES.CMAC.withMACSize(64));
OutputStream sOut = macCalculator.getMACStream();
sOut.write(data);
sOut.close();

return macCalculator.getMAC();

10.3.2 JCE HMAC-SHA256
The following snippet provides a function using the JCE APIs to do HMAC-SHA256 from the passed in HMAC key and data.

```java
public byte[] createMac(SecretKey hmacKey, byte[] data)
    throws Exception
{
    Mac mac = Mac.getInstance("HmacSHA256, "BCFIPS);
    mac.init(hmacKey, name));
    return mac.doFinal(data);
}
```
11 Message Digest Algorithms

The BC FIPS provider supports the full suite of NIST digests up to FIPS PUB-202 (SHA-3), as well
a variety of common ones used by other standards.

11.1 Available in Approved Mode Operation

<table>
<thead>
<tr>
<th>Digest Name</th>
<th>Low-level Class</th>
<th>JCA Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-1</td>
<td>FipsSHS</td>
<td>SHA-1, SHA1</td>
</tr>
<tr>
<td>SHA-224</td>
<td>FipsSHS</td>
<td>SHA-224, SHA224</td>
</tr>
<tr>
<td>SHA-256</td>
<td>FipsSHS</td>
<td>SHA-256, SHA256</td>
</tr>
<tr>
<td>SHA-384</td>
<td>FipsSHS</td>
<td>SHA-384, SHA384</td>
</tr>
<tr>
<td>SHA-512</td>
<td>FipsSHS</td>
<td>SHA-512, SHA512</td>
</tr>
<tr>
<td>SHA-512(224)</td>
<td>FipsSHS</td>
<td>SHA-512(224), SHA512(224)</td>
</tr>
<tr>
<td>SHA-512(256)</td>
<td>FipSHS</td>
<td>SHA-512(256), SHA512(256)</td>
</tr>
<tr>
<td>SHA3-224</td>
<td>FipsSHS</td>
<td>SHA3-224</td>
</tr>
<tr>
<td>SHA3-256</td>
<td>FipsSHS</td>
<td>SHA3-256</td>
</tr>
<tr>
<td>SHA3-384</td>
<td>FipsSHS</td>
<td>SHA3-384</td>
</tr>
<tr>
<td>SHA3-512</td>
<td>FipsSHS</td>
<td>SHA3-512</td>
</tr>
<tr>
<td>SHAKE128</td>
<td>FipsSHS</td>
<td>N/A</td>
</tr>
<tr>
<td>SHAKE256</td>
<td>FipsSHS</td>
<td>N/A</td>
</tr>
</tbody>
</table>

11.2 Available in General Operation

<table>
<thead>
<tr>
<th>Digest Name</th>
<th>Low-level Class</th>
<th>JCA Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOST3411</td>
<td>SecureHash</td>
<td>GOST3411</td>
</tr>
<tr>
<td>RIPEMD128</td>
<td>SecureHash</td>
<td>RIPEMD128</td>
</tr>
<tr>
<td>RIPEMD160</td>
<td>SecureHash</td>
<td>RIPEMD160</td>
</tr>
<tr>
<td>RIPEMD256</td>
<td>SecureHash</td>
<td>RIPEMD256</td>
</tr>
<tr>
<td>RIPEMD320</td>
<td>SecureHash</td>
<td>RIPEMD320</td>
</tr>
<tr>
<td>Tiger</td>
<td>SecureHash</td>
<td>Tiger</td>
</tr>
<tr>
<td>Whirlpool</td>
<td>SecureHash</td>
<td>Whirlpool</td>
</tr>
</tbody>
</table>
11.3 Examples

11.3.1 Use of SHA-256
The following example shows the creation and use of a digest calculator for SHA-256 using a byte array as input.

```java
byte[] input = ...;
FipsDigestOperatorFactory<FipsSHS.Parameters> factory =
    new FipsSHS.OperatorFactory<FipsSHS.Parameters>();
OutputDigestCalculator<FipsSHS.Parameters> calculator =
    factory.createOutputDigestCalculator(FipsSHS.SHA_256);
OutputStream digestStream = calculator.getDigestStream();
digestStream.write(input);
digestStream.close();
byte[] digest = calculator.getDigest();
```

11.3.2 Use of SHAKE128
The following example shows the creation and use of the expandable output function SHAKE128 from the SHA3 family, again using a byte array as input and producing 100 bytes of output.

```java
byte[] input = ...;
FipsXOFOperatorFactory<FipsSHS.Parameters> factory =
    new FipsSHS.XOFOperatorFactory<FipsSHS.Parameters>();
OutputXOFCalculator<FipsSHS.Parameters> calculator =
    factory.createOutputXOFCalculator(FipsSHS.SHAKE128);
OutputStream xofStream = calculator.getFunctionStream();
xofStream.write(input);
xofStream.close();
byte[] output = calculator.getFunctionOutput(100);
```

An interesting feature of the expandable output functions is that you can keep going, hence another call like:

```java
byte[] extra = calculator.getFunctionOutput(100);
```

will just grab another 100 bytes from the function stream. The calculator will keep producing output until either reset() is called or getFunctionStream() is called again (this will trigger a reset as more input will be assumed).

11.3.3 JCA use of SHA3-224
All digests are available using MessageDigest.getInstance() and their JCA name as follows:

```java
byte[] input = ...;
```
MessageDigest digest = MessageDigest.getInstance("SHA3-224", "BCFIPS");

byte[] result = digest.digest(input);
12 Password Based Key Derivation Functions

PBKD functions are supported from the PKCS standards as well as OpenSSL. It is important to note that only PBKDF2 is approved for use in FIPS mode of operation and that this affects the use of things such as PKCS#12 files, which cannot be used in FIPS mode as a result.

12.1 Available in Approved Mode Operation

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Byte Encoding Used</th>
<th>Digest for PRF</th>
<th>Low-level Class</th>
<th>JCE Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBKDF2</td>
<td>UTF-8</td>
<td>SHA-1</td>
<td>FipsPBKD</td>
<td>PBKDF2, PBKDF2withHmacSHA1</td>
</tr>
<tr>
<td>PBKDF2</td>
<td>8-BIT/ASCII</td>
<td>SHA-1</td>
<td>FipsPBKD</td>
<td>PBKDF2with8BIT</td>
</tr>
<tr>
<td>PBKDF2</td>
<td>UTF-8</td>
<td>SHA-224</td>
<td>FipsPBKD</td>
<td>PBKDF2withHmacSHA224</td>
</tr>
<tr>
<td>PBKDF2</td>
<td>UTF-8</td>
<td>SHA-256</td>
<td>FipsPBKD</td>
<td>PBKDF2withHmacSHA256</td>
</tr>
<tr>
<td>PBKDF2</td>
<td>UTF-8</td>
<td>SHA-384</td>
<td>FipsPBKD</td>
<td>PBKDF2withHmacSHA384</td>
</tr>
<tr>
<td>PBKDF2</td>
<td>UTF-8</td>
<td>SHA-512</td>
<td>FipsPBKD</td>
<td>PBKDF2withHmacSHA512</td>
</tr>
</tbody>
</table>

12.2 Available in General Operation

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Byte Encoding Used</th>
<th>Digest for PRF</th>
<th>Low-level Class</th>
<th>JCE Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenSSL</td>
<td>8-BIT/ASCII</td>
<td>MD5</td>
<td>PBKD</td>
<td>PBKDF-OpenSSL</td>
</tr>
<tr>
<td>PBKDF2</td>
<td>UTF-8</td>
<td>GOST-3411</td>
<td>PBKD</td>
<td>PBKDF2withHmacGOST3411</td>
</tr>
<tr>
<td>PBKDF1</td>
<td>8-BIT/ASCII</td>
<td>SHA-1</td>
<td>PBKD</td>
<td>PBEwithSHA1andDES</td>
</tr>
<tr>
<td>PBKDF1</td>
<td>8-BIT/ASCII</td>
<td>MD5</td>
<td>PBKD</td>
<td>PBEwithMD5andDES</td>
</tr>
<tr>
<td>PBKDF1</td>
<td>8-BIT/ASCII</td>
<td>SHA-1</td>
<td>PBKD</td>
<td>PBEwithSHA1andRC2</td>
</tr>
<tr>
<td>PBKDF1</td>
<td>8-BIT/ASCII</td>
<td>MD5</td>
<td>PBKD</td>
<td>PBEwithMD5andRC2</td>
</tr>
<tr>
<td>PKCS#12</td>
<td>PKCS#12</td>
<td>SHA-1</td>
<td>PBKD</td>
<td>PBKDF-PKCS12</td>
</tr>
<tr>
<td>PKCS#12</td>
<td>PKCS#12</td>
<td>SHA-256</td>
<td>PBKD</td>
<td>PBKDF-PKCS12withSHA256</td>
</tr>
<tr>
<td>PKCS#12</td>
<td>PKCS#12</td>
<td>SHA-1</td>
<td>PBKD</td>
<td>PBEwithSHA1and40bitRC4</td>
</tr>
<tr>
<td>PKCS#12</td>
<td>PKCS#12</td>
<td>SHA-1</td>
<td>PBKD</td>
<td>PBEwithSHA1and128bitRC4</td>
</tr>
<tr>
<td>PKCS#12</td>
<td>PKCS#12</td>
<td>SHA-1</td>
<td>PBKD</td>
<td>PBEwithSHA1and40bitRC2</td>
</tr>
<tr>
<td>PKCS#12</td>
<td>PKCS#12</td>
<td>SHA-1</td>
<td>PBKD</td>
<td>PBEwithSHA1and128bitRC2</td>
</tr>
<tr>
<td>PKCS#12</td>
<td>PKCS#12</td>
<td>SHA-1</td>
<td>PBKD</td>
<td>PBEwithSHA1and2-KeyDESede</td>
</tr>
<tr>
<td>PKCS#12</td>
<td>PKCS#12</td>
<td>SHA-1</td>
<td>PBKD</td>
<td>PBEwithSHA1and3-KeyDESede</td>
</tr>
<tr>
<td>PKCS#12</td>
<td>PKCS#12</td>
<td>SHA-1</td>
<td>PBKD</td>
<td>PBEwithSHA1and128BitAES-BC</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Byte Encoding Used</td>
<td>Digest for PRF</td>
<td>Low-level Class</td>
<td>JCE Name</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>PKCS#12</td>
<td>PKCS#12</td>
<td>SHA-1</td>
<td>PBKD</td>
<td>PBEwithSHA1and192BitAES-BC</td>
</tr>
<tr>
<td>PKCS#12</td>
<td>PKCS#12</td>
<td>SHA-1</td>
<td>PBKD</td>
<td>PBEwithSHA1and256BitAES-BC</td>
</tr>
<tr>
<td>PKCS#12</td>
<td>PKCS#12</td>
<td>SHA-256</td>
<td>PBKD</td>
<td>PBEwithSHA256and128BitAES-BC</td>
</tr>
<tr>
<td>PKCS#12</td>
<td>PKCS#12</td>
<td>SHA-256</td>
<td>PBKD</td>
<td>PBEwithSHA256and192BitAES-BC</td>
</tr>
<tr>
<td>PKCS#12</td>
<td>PKCS#12</td>
<td>SHA-256</td>
<td>PBKD</td>
<td>PBEwithSHA256and256BitAES-BC</td>
</tr>
</tbody>
</table>
12.3 Examples

12.3.1 PBKDF2

The following snippet shows the generation of key material using PKCS#5's PBKDF2 function with HMAC SHA-256 being used as the pseudo-random-function underlying the key material generator and password, iterationCount, and salt providing the inputs.

```java
public byte[] getSecretKey(char[] password, int iterationCount, byte[] salt, int keySizeInBits)
{
    PasswordBasedDeriver deriver = new FipsPBKD.DeriverFactory().createDeriver(FipsPBKD.PBKDF2.using(FipsSHS.Algorithm.SHA256_HMAC, PasswordConverter.UTF8.convert(password)).withIterationCount(iterationCount)
        .withSalt(salt));
    return deriver.deriveKey(PasswordBasedDeriver.KeyType.CIPHER, (keySizeInBits + 7) / 8);
}
```

The KeyType parameter above can also be set to KeyType.MAC. In some cases, such as PKCS#12, this will affect the values in the key material that is generated. In the case of PBKDF2 the type parameter is ignored.

12.3.3 PKCS#12

The following snippet generates key and IV material using PKCS#12. Note in this case the derivation function returns an array of byte[], where the first element of the array is the key material and the second element is the IV.

```java
public byte[][] getSecretKeyAndIV(char[] password, int iterationCount, byte[] salt, int keySizeInBits, int ivSizeInBytes)
{
    PasswordBasedDeriver deriver = new PBKD.DeriverFactory().createDeriver(PBKDF.PKCS12.using(PasswordConverter.PKCS12, password).withIterationCount(iterationCount).withSalt(salt));
    return deriver.deriveKeyAndIV(PasswordBasedDeriver.KeyType.CIPHER, (keySizeInBits + 7) / 8, ivSizeInBytes);
}
```

12.3.3 JCE PBKDF2

The following examples shows the same function as described in 11.3.1, except in this case it is defined in the context of the JCE.

```java
public SecretKey getSecretKey(char[] password, int iterationCount, byte[] salt, int keySizeInBits)
{
    SecretKeyFactory keyFact = SecretKeyFactory.getInstance("PBKDF2WITHHMACSHA256", "BCFIPS");
    PBEKeySpec pbeSpec = new PBEKeySpec(password, salt, iterationCount, keySizeInBits);
    return keyFact.generateSecret(pbeSpec);
}
```
13 Random Number Generators

For the most part SecureRandom objects need to be created using the low-level class as the JCA does not provide the ability to configure them and this is required for both FIPS and X9.31.

13.1 Available in Approved Mode Operation

The following DRBG types can be constructed in approved mode of operation.

<table>
<thead>
<tr>
<th>DRBG Name</th>
<th>Low-level Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>HASH DRBG</td>
<td>FipsDRBG</td>
</tr>
<tr>
<td>HMAC DRBG</td>
<td>FipsDRBG</td>
</tr>
<tr>
<td>CTR DRBG</td>
<td>FipsDRBG</td>
</tr>
</tbody>
</table>

The BouncyCastleFipsProvider also makes available two SecureRandom which are based on the provider's configuration and use approved mode DRBGs. You can get access to the provider SecureRandom using the name “DEFAULT”, as in:

SecureRandom random = SecureRandom.getInstance("DEFAULT", "BCFIPS");

The “DEFAULT” SecureRandom is configured to be prediction resistant.

The second SecureRandom is the “NONCEANDIV” SecureRandom which is suitable for nonce and IV material. The “NONCEANDIV” SecureRandom can be accessed by:

SecureRandom ivRandom = SecureRandom.getInstance("NONCEANDIV", "BCFIPS");

13.2 Available in General Operation

The X9.31 PRNG is available in general operation, both Triple-DES and AES are supported.

<table>
<thead>
<tr>
<th>DRBG Name</th>
<th>Low-level Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>X9.31</td>
<td>X931PRNG</td>
</tr>
</tbody>
</table>

13.3 Examples

13.3.1 Creation for SHA512 DRBG

This example creates a DRBG based on a default SecureRandom, in this case the DRBG will reseed itself using the generateSeed() method on the SecureRandom, and the nonce is generated using one of the recommended techniques in NIST SP 800-90A, a string of bits from the entropy source of \( \frac{1}{2} \) the security strength in size.

byte[] personalizationString = Strings.toUTF8ByteArray(new VMID().toString());

SecureRandom entropySource = new SecureRandom();

SecureRandom random = FipsDRBG.SHA512.fromEntropySource(entropySource, true)
.setPersonalizationString(personalizationString)
13.3.2 Creation for AES X9.31
The following example creates a X9.31 PRNG base on a default SecureRandom.

```java
X931PRNG.Builder bld = X931PRNG.AES_128.fromEntropySource(new SecureRandom(), true);
bld.setDateTimeVector(Hex.decode("259e67249288597a4d61e7c0e690afae"));
SecureRandom rand = bld.build(new SymmetricSecretKey(X931PRNG.AES_128,
    Hex.decode("f7d36762b9915f1ed585e8e91700eb2")), false);
```
14 Signature Algorithms

A range of signature algorithms are available both in the low-level API and the provider. Some non-FIPS variations of RSA, DSA, and ECDSA are also available in the general operation set.

14.1 Available in Approved Mode Operation

In approved mode of operation DSA keys of 1024 bits are supported for signature verification only.

14.1.1 DSA

<table>
<thead>
<tr>
<th>Signature</th>
<th>Low-level Class</th>
<th>JCA Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA1 with DSA</td>
<td>FipsDSA</td>
<td>SHA1withDSA</td>
</tr>
<tr>
<td>SHA224 with DSA</td>
<td>FipsDSA</td>
<td>SHA224withDSA</td>
</tr>
<tr>
<td>SHA256 with DSA</td>
<td>FipsDSA</td>
<td>SHA256withDSA</td>
</tr>
<tr>
<td>SHA384 with DSA</td>
<td>FipsDSA</td>
<td>SHA384withDSA</td>
</tr>
<tr>
<td>SHA512 with DSA</td>
<td>FipsDSA</td>
<td>SHA512withDSA</td>
</tr>
<tr>
<td>SHA512(224) with DSA</td>
<td>FipsDSA</td>
<td>SHA512(224)withDSA</td>
</tr>
<tr>
<td>SHA512(256) with DSA</td>
<td>FipsDSA</td>
<td>SHA512(256)withDSA</td>
</tr>
</tbody>
</table>

14.1.2 EC

<table>
<thead>
<tr>
<th>Signature</th>
<th>Low-level Class</th>
<th>JCA Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA1 with ECDSA</td>
<td>FipsEC</td>
<td>SHA1withECDSA</td>
</tr>
<tr>
<td>SHA224 with ECDSA</td>
<td>FipsEC</td>
<td>SHA224withECDSA</td>
</tr>
<tr>
<td>SHA256 with ECDSA</td>
<td>FipsEC</td>
<td>SHA256withECDSA</td>
</tr>
<tr>
<td>SHA384 with ECDSA</td>
<td>FipsEC</td>
<td>SHA384withECDSA</td>
</tr>
<tr>
<td>SHA512 with ECDSA</td>
<td>FipsEC</td>
<td>SHA512withECDSA</td>
</tr>
<tr>
<td>SHA512(224) with ECDSA</td>
<td>FipsEC</td>
<td>SHA512(224)withECDSA</td>
</tr>
<tr>
<td>SHA512(256) with ECDSA</td>
<td>FipsEC</td>
<td>SHA512(256)withECDSA</td>
</tr>
</tbody>
</table>

14.1.3 RSA

In approved mode of operation RSA keys of 1024 bit are supported for signature verification only.

14.1.3.1 PKCS1.5
<table>
<thead>
<tr>
<th>Signature</th>
<th>Low-level Class</th>
<th>JCA Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA1 with RSA</td>
<td>FipsRSA</td>
<td>SHA1withRSA</td>
</tr>
<tr>
<td>SHA224 with RSA</td>
<td>FipsRSA</td>
<td>SHA224withRSA</td>
</tr>
<tr>
<td>SHA256 with RSA</td>
<td>FipsRSA</td>
<td>SHA256withRSA</td>
</tr>
<tr>
<td>SHA384 with RSA</td>
<td>FipsRSA</td>
<td>SHA384withRSA</td>
</tr>
<tr>
<td>SHA512 with RSA</td>
<td>FipsRSA</td>
<td>SHA512withRSA</td>
</tr>
<tr>
<td>SHA512(224) with RSA</td>
<td>FipsRSA</td>
<td>SHA512(224)withRSA</td>
</tr>
<tr>
<td>SHA512(256) with RSA</td>
<td>FipsRSA</td>
<td>SHA512(256)withRSA</td>
</tr>
</tbody>
</table>

14.1.3.2 PSS

<table>
<thead>
<tr>
<th>Signature</th>
<th>Low-level Class</th>
<th>JCA Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSS SHA1 with RSA</td>
<td>FipsRSA</td>
<td>SHA1withRSAandMGF1, SHA1withRSA/PSS</td>
</tr>
<tr>
<td>PSS SHA224 with RSA</td>
<td>FipsRSA</td>
<td>SHA224withRSAandMGF1, SHA224withRSA/PSS</td>
</tr>
<tr>
<td>PSS SHA256 with RSA</td>
<td>FipsRSA</td>
<td>SHA256withRSAandMGF1, SHA256withRSA/PSS</td>
</tr>
<tr>
<td>PSS SHA384 with RSA</td>
<td>FipsRSA</td>
<td>SHA384withRSAandMGF1, SHA384withRSA/PSS</td>
</tr>
<tr>
<td>PSS SHA512 with RSA</td>
<td>FipsRSA</td>
<td>SHA512withRSAandMGF1, SHA512withRSA/PSS</td>
</tr>
<tr>
<td>PSS SHA512(224) with RSA</td>
<td>FipsRSA</td>
<td>SHA512(224)withRSAandMGF1, SHA512(224)withRSA/PSS</td>
</tr>
<tr>
<td>PSS SHA512(256) with RSA</td>
<td>FipsRSA</td>
<td>SHA512(256)withRSAandMGF1, SHA512(256)withRSA/PSS</td>
</tr>
</tbody>
</table>

14.1.3.3 X9.31

<table>
<thead>
<tr>
<th>Signature</th>
<th>Low-level Class</th>
<th>JCA Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>X9.31 SHA1 with RSA</td>
<td>FipsRSA</td>
<td>SHA1withRSA/X9.31</td>
</tr>
<tr>
<td>X9.31 SHA224 with RSA</td>
<td>FipsRSA</td>
<td>SHA224withRSA/X9.31</td>
</tr>
<tr>
<td>X9.31 SHA256 with RSA</td>
<td>FipsRSA</td>
<td>SHA256withRSA/X9.31</td>
</tr>
<tr>
<td>X9.31 SHA384 with RSA</td>
<td>FipsRSA</td>
<td>SHA384withRSA/X9.31</td>
</tr>
<tr>
<td>X9.31 SHA512 with RSA</td>
<td>FipsRSA</td>
<td>SHA512withRSA/X9.31</td>
</tr>
<tr>
<td>X9.31 SHA512(224) with RSA</td>
<td>FipsRSA</td>
<td>SHA512(224)withRSA/X9.31</td>
</tr>
<tr>
<td>X9.31 SHA512(256) with RSA</td>
<td>FipsRSA</td>
<td>SHA512(256)withRSA/X9.31</td>
</tr>
</tbody>
</table>
14.2 Available in General Operation

14.2.1 DSA

14.2.1.1 Regular DSA
There are currently no extra JCA DSA signature types available in general mode operation.

14.2.1.2 Deterministic DSA

<table>
<thead>
<tr>
<th>Signature</th>
<th>Low-level Class</th>
<th>JCA Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deterministic SHA1 with DSA</td>
<td>DSA</td>
<td>SHA1withDDSA</td>
</tr>
<tr>
<td>Deterministic SHA224 with DSA</td>
<td>DSA</td>
<td>SHA224withDDSA</td>
</tr>
<tr>
<td>Deterministic SHA256 with DSA</td>
<td>DSA</td>
<td>SHA256withDDSA</td>
</tr>
<tr>
<td>Deterministic SHA384 with DSA</td>
<td>DSA</td>
<td>SHA384withDDSA</td>
</tr>
<tr>
<td>Deterministic SHA512 with DSA</td>
<td>DSA</td>
<td>SHA512withDDSA</td>
</tr>
<tr>
<td>Deterministic SHA512(224) with DSA</td>
<td>DSA</td>
<td>SHA512(224)withDDSA</td>
</tr>
<tr>
<td>Deterministic SHA512(256) with DSA</td>
<td>DSA</td>
<td>SHA512(256)withDDSA</td>
</tr>
</tbody>
</table>

14.2.2 DSTU4145

<table>
<thead>
<tr>
<th>Signature</th>
<th>Low-level Class</th>
<th>JCA Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOST3411 with DSTU4145</td>
<td>DSTU4145</td>
<td>GOST3411withDSTU4145</td>
</tr>
<tr>
<td>GOST3411 with DSTU4145 little endian</td>
<td>DSTU4145</td>
<td>GOST3411withDSTU4145LE</td>
</tr>
</tbody>
</table>

14.2.3 ECDSA

14.2.3.1 Regular ECDSA

<table>
<thead>
<tr>
<th>Signature</th>
<th>Low-level Class</th>
<th>JCA Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>RipeMD160 with ECDSA</td>
<td>EC</td>
<td>RipeMD160withECDSA</td>
</tr>
</tbody>
</table>

14.2.3.2 Deterministic ECDSA

<table>
<thead>
<tr>
<th>Signature</th>
<th>Low-level Class</th>
<th>JCA Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deterministic SHA1 with ECDSA</td>
<td>EC</td>
<td>SHA1withECDDDSA</td>
</tr>
<tr>
<td>Deterministic SHA224 with ECDSA</td>
<td>EC</td>
<td>SHA224withECDDDSA</td>
</tr>
<tr>
<td>Deterministic SHA256 with ECDSA</td>
<td>EC</td>
<td>SHA256withECDDDSA</td>
</tr>
<tr>
<td>Deterministic SHA384 with ECDSA</td>
<td>EC</td>
<td>SHA384withECDDDSA</td>
</tr>
<tr>
<td>Signature</td>
<td>Low-level Class</td>
<td>JCA Name</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Deterministic SHA512 with ECDSA</td>
<td>EC</td>
<td>SHA512withECDDSA</td>
</tr>
<tr>
<td>Deterministic SHA512 with ECDSA</td>
<td>EC</td>
<td>SHA512(224)withECDDSA</td>
</tr>
<tr>
<td>Deterministic SHA512 with ECDSA</td>
<td>EC</td>
<td>SHA512(256)withECDDSA</td>
</tr>
</tbody>
</table>

### 14.2.4 GOST3410-1994

<table>
<thead>
<tr>
<th>Signature</th>
<th>Low-level Class</th>
<th>JCA Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOST3411 with GOST3410-1994</td>
<td>GOST3410</td>
<td>GOST3411withGOST3410</td>
</tr>
</tbody>
</table>

### 14.2.5 GOST3410-2001

<table>
<thead>
<tr>
<th>Signature</th>
<th>Low-level Class</th>
<th>JCA Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOST3411 with GOST3410-2001</td>
<td>ECGOST3410</td>
<td>GOST3411withECGOST3410</td>
</tr>
</tbody>
</table>

### 14.2.6 RSA

#### 14.2.6.1 ISO9796-2

<table>
<thead>
<tr>
<th>Signature</th>
<th>Low-level Class</th>
<th>JCA Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISO9796-2 SHA1 with RSA</td>
<td>RSA</td>
<td>SHA1withRSA/ISO9796-2</td>
</tr>
<tr>
<td>ISO9796-2 SHA224 with RSA</td>
<td>RSA</td>
<td>SHA224withRSA/ISO9796-2</td>
</tr>
<tr>
<td>ISO9796-2 SHA256 with RSA</td>
<td>RSA</td>
<td>SHA256withRSA/ISO9796-2</td>
</tr>
<tr>
<td>ISO9796-2 SHA384 with RSA</td>
<td>RSA</td>
<td>SHA384withRSA/ISO9796-2</td>
</tr>
<tr>
<td>ISO9796-2 SHA512 with RSA</td>
<td>RSA</td>
<td>SHA512withRSA/ISO9796-2</td>
</tr>
<tr>
<td>ISO9796-2 SHA512(224) with RSA</td>
<td>RSA</td>
<td>SHA512(224)withRSA/ISO9796-2</td>
</tr>
<tr>
<td>ISO9796-2 SHA512(256) with RSA</td>
<td>RSA</td>
<td>SHA512(256)withRSA/ISO9796-2</td>
</tr>
<tr>
<td>ISO9796-2 RIPEMD128 with RSA</td>
<td>RSA</td>
<td>RIPEMD128withRSA/ISO9796-2</td>
</tr>
<tr>
<td>ISO9796-2 RIPEMD160 with RSA</td>
<td>RSA</td>
<td>RIPEMD160withRSA/ISO9796-2</td>
</tr>
</tbody>
</table>

#### 14.2.6.2 ISO9796-2/PSS

<table>
<thead>
<tr>
<th>Signature</th>
<th>Low-level Class</th>
<th>JCA Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISO9796-2 PSS SHA1 with RSA</td>
<td>RSA</td>
<td>SHA1withRSA/ISO9796-2PSS</td>
</tr>
<tr>
<td>ISO9796-2 PSS SHA224 with RSA</td>
<td>RSA</td>
<td>SHA224withRSA/ISO9796-2PSS</td>
</tr>
<tr>
<td>Signature</td>
<td>Low-level Class</td>
<td>JCA Name</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>ISO9796-2 PSS SHA256 with RSA</td>
<td>RSA</td>
<td>SHA256withRSA/ISO9796-2PSS</td>
</tr>
<tr>
<td>ISO9796-2 PSS SHA384 with RSA</td>
<td>RSA</td>
<td>SHA384withRSA/ISO9796-2PSS</td>
</tr>
<tr>
<td>ISO9796-2 PSS SHA512 with RSA</td>
<td>RSA</td>
<td>SHA512withRSA/ISO9796-2PSS</td>
</tr>
<tr>
<td>ISO9796-2 PSS SHA512(224) with RSA</td>
<td>RSA</td>
<td>SHA512(224)withRSA/ISO9796-2PSS</td>
</tr>
<tr>
<td>ISO9796-2 PSS SHA512(256) with RSA</td>
<td>RSA</td>
<td>SHA512(256)withRSA/ISO9796-2PSS</td>
</tr>
<tr>
<td>ISO9796-2 PSS RIPEMD128 with RSA</td>
<td>RSA</td>
<td>RIPEMD128withRSA/ISO9796-2PSS</td>
</tr>
<tr>
<td>ISO9796-2 PSS RIPEMD160 with RSA</td>
<td>RSA</td>
<td>RIPEMD160withRSA/ISO9796-2PSS</td>
</tr>
</tbody>
</table>

### 14.2.6.3 PKCS1.5

<table>
<thead>
<tr>
<th>Signature</th>
<th>Low-level Class</th>
<th>JCA Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5 with RSA</td>
<td>RSA</td>
<td>MD5withRSA</td>
</tr>
<tr>
<td>RIPEMD128 with RSA</td>
<td>RSA</td>
<td>RIPEMD128withRSA</td>
</tr>
<tr>
<td>RIPEMD160 with RSA</td>
<td>RSA</td>
<td>RIPEMD160withRSA</td>
</tr>
<tr>
<td>RIPEMD256 with RSA</td>
<td>RSA</td>
<td>RIPEMD256withRSA</td>
</tr>
</tbody>
</table>

### 14.2.6.4 PSS

There are currently no extra JCA PSS signature types available in general mode operation.

### 14.2.6.5 X9.31

<table>
<thead>
<tr>
<th>Signature</th>
<th>Low-level Class</th>
<th>JCA Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>X9.31 RIPEMD128 with RSA</td>
<td>RSA</td>
<td>RIPEMD128withRSA/X9.31</td>
</tr>
<tr>
<td>X9.31 RIPEMD160 with RSA</td>
<td>RSA</td>
<td>RIPEMD160withRSA/X9.31</td>
</tr>
<tr>
<td>X9.31 Whirlpool with RSA</td>
<td>RSA</td>
<td>WhirlpoolwithRSA/X9.31</td>
</tr>
</tbody>
</table>

### 14.3 Examples

#### 14.3.1 RSA with SHA-1

The following sample shows a method for generating a signature in PKCS#1 v1.5 format.

Note: while PKCS#1 v1.5 signing does not use random data in the actual signature calculation, a SecureRandom is required in this case to provide blinding to help protect the private key.

```java
public byte[] signData(AsymmetricRSAPrivateKey privKey, byte[] data) throws Exception {
```
FipsRSA.SignatureOperatorFactory signatureOperatorFactory =
    new FipsRSA.SignatureOperatorFactory();

OutputStream sOut = rsaSig.getSigningStream();
sOut.write(data, 0, data.length);
sOut.close();
return rsaSig.getSignature();
}

The following method can be used to verify the signature return by signData providing pubKey is
the public key corresponding to privKey used in the signData method.

public boolean isVerified(
    AsymmetricRSAPublicKey pubKey, byte[] signature, byte[] data)
 throws Exception
{
    FipsRSA.SignatureOperatorFactory signatureOperatorFactory =
        new FipsRSA.SignatureOperatorFactory();

    OutputStream vOut = rsaVer.getVerifyingStream();
vOut.write(data, 0, data.length);
vOut.close();
    return rsaVer.isVerified(signature);
}

14.3.2 JCA ECDSA with SHA-256

The following sample shows a method for generating an ECDSA signature with SHA-256 using the
JCA provider. This method will use the provider default SecureRandom for generating the random
component for the ECDSA signature.

public byte[] signData(ECPrivateKey privKey, byte[] data)
 throws Exception
{
    Signature sig = Signature.getInstance("SHA256withECDSA", "BCFIPS");
    sig.initSign(privKey);
    sig.update(dummySha1);
    return sig.sign();
}

The following method can be used to verify the signature return by signData providing pubKey is
the public key corresponding to privKey used in the signData method.
public boolean isVerified(ECPublicKey pubKey, byte[] signature, byte[] data)
    throws Exception
{
    Signature sig = Signature.getInstance("SHA256withECDSA", "BCFIPS");
    sig.initVerify(pubKey);
    sig.update(data);
    return sig.verify((signature);
}
Appendix A – System Properties

By default all the below properties are assumed to be false.

org.bouncycastle.fips.approved_only – if this property is set to true the module will start in approved mode and non-approved mode functionality will not be available. Note: to be effective this property needs to be set for the JVM, or before the CryptoServicesRegistrar class has loaded.

org.bouncycastle.rsa.allow_multi_use – in approved/unapproved mode the module will attempt to block an RSA modulus from being used for encryption if it has been used for signing, or visa-versa. It is possible to stop this from happening by setting org.bouncycastle.rsa.allow_multi_use to true.

org.bouncycastle.dsa.FIPS186-2for1024bits – this property only has an effect in unapproved mode. If legacy DSA parameters must be generated and the parameter size is 1024 setting this property to true will result in the FIPS 186-2 algorithm being used for parameter generation.

org.bouncycastle.jsse.disable_kdf – the BCFIPS module provides its own interpretation of the standard JSSE KDF generators. At the time of writing the parameters for these are mainly defined using internal classes, which may change. Setting this property to true will result in the BouncyCastleFipsProvider not making the JSSE KDFs available which will cause the JSSE to fall back to its own KDFs if it is not running in FIPS mode.

org.bouncycastle.pkix.disable_certpath – in some cases it's easier to use the default CertPath implementation, rather than the one provided in the module. Setting this property to true will result in the BCFIPS CertPathValidator and CertPathBuilder not being created in an BouncyCastleFipsProvider that is instanced after the property is set.

org.bouncycastle.trpaper.allow_weak – setting this property to true will allow the use of TripleDES weak keys. This is only present as it is a requirement for CAVP testing.

org.bouncycastle.ec.disable_mqv – setting this property to true will disable support for EC MQV.

org.bouncycastle.pkcs1.not_strict – some other providers of cryptography services fail to produce PKCS1 encoded block that are the correct length. Setting this property to true will relax the conformance check on the block length.

org.bouncycastle.jca.enable_jks – enable the JKS key store for FIPS. In order to maintain compliance this key store can only be used for reading JKS keystores containing certificates. Default is false to prevent unnecessary conflicts with the regular Java one. If set to true the key store is enabled.
Appendix B – Policy Permissions

If the BC FIPS module is used in association with a SecurityManager there are some Java permissions that need to be set in applications policy file, together with some optional ones that are specific to the BC FIPS module.

B.1 Java Permissions

In order to do the checksum validation of the jar

```java
permission java.lang.RuntimePermission "getProtectionDomain";
```

needs to be enabled in order for the module jar to examine its own contents.

In order to check for configuration properties the policy permission:

```java
permission java.util.PropertyPermission "java.runtime.name", "read";
```

also need to be provided.

The module also makes use of reflection to enable use of later than JDK 1.5 classes. In order to enable this the policy permission:

```java
permission java.lang.RuntimePermission "accessDeclaredMembers";
```

is required.

If the JCA/JCE provider is to be installed during execution, the policy permission:

```java
permission java.security.SecurityPermission "putProviderProperty.BCFIPS";
```

is also required.

B.2 Optional Permissions

B.2.1 Configuration of Approved/Unapproved Modes

CryptoServicesRegistrar calculates the default mode of operation based on the granting of

```java
permission org.bouncycastle.crypto.CryptoServicesPermission “unapprovedModeEnabled”;
```

If this permission is granted by the security manager, then the JVM will start threads in a default of unapproved mode.

If this permission is not granted by the security manager, then the JVM will start threads in the approved mode only.
B.2.2 Use of CryptoServicesRegistrar.setApprovedMode(true)

If the JVM has been granted the use of unapproved mode services then a thread may move into approved mode by calling CryptoServicesRegistrar.setApprovedMode(true) if the permission:

```
permission
org.bouncycastle.crypto.CryptoServicesPermission "changeToApprovedModeEnabled";
```

is granted.

If the permission is not granted together and a thread is not already in approved mode then the call to CryptoServicesRegistrar.setApprovedMode(true) will result in an exception being thrown.

B.2.3 Key Export and Translation

```
permission org.bouncycastle.crypto.CryptoServicesPermission "exportSecretKey";
and
permission org.bouncycastle.crypto.CryptoServicesPermission "exportPrivateKey";
```

or

```
permission org.bouncycastle.crypto.CryptoServicesPermission "exportKeys";
```

are required to do any exporting of CSPs outside of the module. These permissions are also required to be set to allow repackaging of keys between layers.

If neither of these permissions are set it is possible to import keys into the module and to generate keys within it, however without them the private values can never be displayed or persisted.

B.2.4 SSL Support

RSA PKCS#1.5 key wrap, NONEwithDSA, NONEwithECDSA, and NONEwithRSA require 2 additional policy settings if the BCFIPS provider is run with a SecurityManager present and in “FIPS only” mode – as a rule these algorithms are not FIPS approved, except where used for TLS and the policy settings reflect this. In the most general case this will need:

```
permission
org.bouncycastle.crypto.CryptoServicesPermission "tlsAlgorithmsEnabled";
```

which is the same as setting the following two permissions together:

```
permission
org.bouncycastle.crypto.CryptoServicesPermission "tlsNullDigestEnabled";
```

```
permission
org.bouncycastle.crypto.CryptoServicesPermission "tlsPKCS15KeyWrapEnabled";
```
B.2.5 Setting of Default SecureRandom

Permission is required for threads to set the default secure random in the presence of a security manager.

The following permission is required to set the default secure random using the CryptoServicesRegistrar

```
permission
    org.bouncycastle.crypto.CryptoServicesPermission "defaultRandomConfig";
```

B.2.6 Setting CryptoServicesRegistrar Properties

Permissions are required for threads to either set thread local properties or global properties in the CryptoServicesRegistrar. Possession of permission to set a global property in the CryptoServicesRegistrar automatically implies permission to set a thread local property.

The following permission is required to set a thread local property on the CryptoServicesRegistrar

```
permission
    org.bouncycastle.crypto.CryptoServicesPermission "threadLocalConfig";
```

The following permission is required to set a global property on the CryptoServicesRegistrar

```
permission
    org.bouncycastle.crypto.CryptoServicesPermission "globalConfig";
```
Appendix C – Built in Curves

The following curves are available in general mode operation for ECDSA and/or ECDH/ECCDH/ECMQV. In the case of approved only mode, only curves offering a security level of 112 bits or greater can be used.

<table>
<thead>
<tr>
<th>Name</th>
<th>IDs</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSSI FRP 256v1</td>
<td>FRP256v1</td>
<td>1.2.250.1.223.101.256.1</td>
</tr>
<tr>
<td>ECC Brainpool P160r1</td>
<td>brainpoolp160r1</td>
<td>1.3.36.3.3.2.8.1.1.1</td>
</tr>
<tr>
<td>ECC Brainpool P160t1</td>
<td>brainpoolp160t1</td>
<td>1.3.36.3.3.2.8.1.1.2</td>
</tr>
<tr>
<td>ECC Brainpool P192r1</td>
<td>brainpoolp192r1</td>
<td>1.3.36.3.3.2.8.1.1.3</td>
</tr>
<tr>
<td>ECC Brainpool P192t1</td>
<td>brainpoolp192t1</td>
<td>1.3.36.3.3.2.8.1.1.4</td>
</tr>
<tr>
<td>ECC Brainpool P224r1</td>
<td>brainpoolp224r1</td>
<td>1.3.36.3.3.2.8.1.1.5</td>
</tr>
<tr>
<td>ECC Brainpool P224t1</td>
<td>brainpoolp224t1</td>
<td>1.3.36.3.3.2.8.1.1.6</td>
</tr>
<tr>
<td>ECC Brainpool P256r1</td>
<td>brainpoolp256r1</td>
<td>1.3.36.3.3.2.8.1.1.7</td>
</tr>
<tr>
<td>ECC Brainpool P256t1</td>
<td>brainpoolp256t1</td>
<td>1.3.36.3.3.2.8.1.1.8</td>
</tr>
<tr>
<td>ECC Brainpool P320r1</td>
<td>brainpoolp320r1</td>
<td>1.3.36.3.3.2.8.1.1.9</td>
</tr>
<tr>
<td>ECC Brainpool P320t1</td>
<td>brainpoolp320t1</td>
<td>1.3.36.3.3.2.8.1.1.10</td>
</tr>
<tr>
<td>ECC Brainpool P384r1</td>
<td>brainpoolp384r1</td>
<td>1.3.36.3.3.2.8.1.1.11</td>
</tr>
<tr>
<td>ECC Brainpool P384t1</td>
<td>brainpoolp384t1</td>
<td>1.3.36.3.3.2.8.1.1.12</td>
</tr>
<tr>
<td>ECC Brainpool P512r1</td>
<td>brainpoolp512r1</td>
<td>1.3.36.3.3.2.8.1.1.13</td>
</tr>
<tr>
<td>ECC Brainpool P512t1</td>
<td>brainpoolp512t1</td>
<td>1.3.36.3.3.2.8.1.1.14</td>
</tr>
<tr>
<td>NIST B-163</td>
<td>B-163</td>
<td>1.3.132.0.15</td>
</tr>
<tr>
<td>NIST B-233</td>
<td>B-233</td>
<td>1.3.132.0.27</td>
</tr>
<tr>
<td>NIST B-283</td>
<td>B-283</td>
<td>1.3.132.0.17</td>
</tr>
<tr>
<td>NIST B-409</td>
<td>B-409</td>
<td>1.3.132.0.37</td>
</tr>
<tr>
<td>NIST B-571</td>
<td>B-571</td>
<td>1.3.132.0.39</td>
</tr>
<tr>
<td>NIST K-163</td>
<td>K-163</td>
<td>1.3.132.0.1</td>
</tr>
<tr>
<td>NIST K-233</td>
<td>K-233</td>
<td>1.3.132.0.26</td>
</tr>
<tr>
<td>NIST K-283</td>
<td>K-283</td>
<td>1.3.132.0.16</td>
</tr>
<tr>
<td>NIST K-409</td>
<td>K-409</td>
<td>1.3.132.0.36</td>
</tr>
<tr>
<td>NIST K-571</td>
<td>K-571</td>
<td>1.3.132.0.38</td>
</tr>
<tr>
<td>NIST P-192</td>
<td>P-192</td>
<td>1.2.840.10045.3.1.1</td>
</tr>
<tr>
<td>NIST P-224</td>
<td>P-224</td>
<td>1.3.132.0.33</td>
</tr>
<tr>
<td>NIST P-256</td>
<td>P-256</td>
<td>1.2.840.10045.3.1.7</td>
</tr>
<tr>
<td>NIST P-384</td>
<td>P-384</td>
<td>1.3.132.0.34</td>
</tr>
<tr>
<td>NIST P-521</td>
<td>P-521</td>
<td>1.3.132.0.35</td>
</tr>
<tr>
<td>SEC secp112r1</td>
<td>secp112r1</td>
<td>1.3.132.0.6</td>
</tr>
<tr>
<td>Name</td>
<td>IDs</td>
<td>OID</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------</td>
<td>-------------------</td>
</tr>
<tr>
<td>SEC secp112r2</td>
<td>secp112r2</td>
<td>1.3.132.0.7</td>
</tr>
<tr>
<td>SEC secp128r1</td>
<td>secp128r1</td>
<td>1.3.132.0.28</td>
</tr>
<tr>
<td>SEC secp128r2</td>
<td>secp128r2</td>
<td>1.3.132.0.29</td>
</tr>
<tr>
<td>SEC secp160k1</td>
<td>secp160k1</td>
<td>1.3.132.0.9</td>
</tr>
<tr>
<td>SEC secp160r1</td>
<td>secp160r1</td>
<td>1.3.132.0.8</td>
</tr>
<tr>
<td>SEC secp160r2</td>
<td>secp160r2</td>
<td>1.3.132.0.30</td>
</tr>
<tr>
<td>SEC secp192k1</td>
<td>secp192k1</td>
<td>1.3.132.0.31</td>
</tr>
<tr>
<td>SEC secp192r1</td>
<td>secp192r1</td>
<td>1.2.840.10045.3.1.1</td>
</tr>
<tr>
<td>SEC secp224k1</td>
<td>secp224k1</td>
<td>1.3.132.0.32</td>
</tr>
<tr>
<td>SEC secp224r1</td>
<td>secp224r1</td>
<td>1.3.132.0.33</td>
</tr>
<tr>
<td>SEC secp256k1</td>
<td>secp256k1</td>
<td>1.3.132.0.10</td>
</tr>
<tr>
<td>SEC secp256r1</td>
<td>secp256r1</td>
<td>1.2.840.10045.3.1.7</td>
</tr>
<tr>
<td>SEC secp384r1</td>
<td>secp384r1</td>
<td>1.3.132.0.34</td>
</tr>
<tr>
<td>SEC secp521r1</td>
<td>secp521r1</td>
<td>1.3.132.0.35</td>
</tr>
<tr>
<td>SEC sect113r1</td>
<td>sect113r1</td>
<td>1.3.132.0.4</td>
</tr>
<tr>
<td>SEC sect113r2</td>
<td>sect113r2</td>
<td>1.3.132.0.5</td>
</tr>
<tr>
<td>SEC sect131r1</td>
<td>sect131r1</td>
<td>1.3.132.0.22</td>
</tr>
<tr>
<td>SEC sect131r2</td>
<td>sect131r2</td>
<td>1.3.132.0.23</td>
</tr>
<tr>
<td>SEC sect163k1</td>
<td>sect163k1</td>
<td>1.3.132.0.1</td>
</tr>
<tr>
<td>SEC sect163r1</td>
<td>sect163r1</td>
<td>1.3.132.0.2</td>
</tr>
<tr>
<td>SEC sect163r2</td>
<td>sect163r2</td>
<td>1.3.132.0.15</td>
</tr>
<tr>
<td>SEC sect193r1</td>
<td>sect193r1</td>
<td>1.3.132.0.24</td>
</tr>
<tr>
<td>SEC sect193r2</td>
<td>sect193r2</td>
<td>1.3.132.0.25</td>
</tr>
<tr>
<td>SEC sect233k1</td>
<td>sect233k1</td>
<td>1.3.132.0.26</td>
</tr>
<tr>
<td>SEC sect233r1</td>
<td>sect233r1</td>
<td>1.3.132.0.27</td>
</tr>
<tr>
<td>SEC sect239k1</td>
<td>sect239k1</td>
<td>1.3.132.0.3</td>
</tr>
<tr>
<td>SEC sect283k1</td>
<td>sect283k1</td>
<td>1.3.132.0.16</td>
</tr>
<tr>
<td>SEC sect409k1</td>
<td>sect409k1</td>
<td>1.3.132.0.36</td>
</tr>
<tr>
<td>SEC sect409r1</td>
<td>sect409r1</td>
<td>1.3.132.0.37</td>
</tr>
<tr>
<td>SEC sect571k1</td>
<td>sect571k1</td>
<td>1.3.132.0.38</td>
</tr>
<tr>
<td>SEC sect571r1</td>
<td>sect571r1</td>
<td>1.3.132.0.39</td>
</tr>
<tr>
<td>X9.62 c2pnb163v1</td>
<td>c2pnb163v1</td>
<td>1.2.840.10045.3.0.1</td>
</tr>
<tr>
<td>X9.62 c2pnb163v2</td>
<td>c2pnb163v2</td>
<td>1.2.840.10045.3.0.2</td>
</tr>
<tr>
<td>X9.62 c2pnb163v3</td>
<td>c2pnb163v3</td>
<td>1.2.840.10045.3.0.3</td>
</tr>
<tr>
<td>X9.62 c2pnb176w1</td>
<td>c2pnb176w1</td>
<td>1.2.840.10045.3.0.4</td>
</tr>
<tr>
<td>X9.62 c2tmbr191v1</td>
<td>c2tmbr191v1</td>
<td>1.2.840.10045.3.0.5</td>
</tr>
<tr>
<td>Name</td>
<td>IDs</td>
<td>OID</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>X9.62 c2tnb191v2</td>
<td>c2tnb191v2</td>
<td>1.2.840.10045.3.0.6</td>
</tr>
<tr>
<td>X9.62 c2tnb191v3</td>
<td>c2tnb191v3</td>
<td>1.2.840.10045.3.0.7</td>
</tr>
<tr>
<td>X9.62 c2pnb208w1</td>
<td>c2pnb208w1</td>
<td>1.2.840.10045.3.0.10</td>
</tr>
<tr>
<td>X9.62 c2tnb239v1</td>
<td>c2tnb239v1</td>
<td>1.2.840.10045.3.0.11</td>
</tr>
<tr>
<td>X9.62 c2tnb239v2</td>
<td>c2tnb239v2</td>
<td>1.2.840.10045.3.0.12</td>
</tr>
<tr>
<td>X9.62 c2tnb239v3</td>
<td>c2tnb239v3</td>
<td>1.2.840.10045.3.0.13</td>
</tr>
<tr>
<td>X9.62 c2pnb272w1</td>
<td>c2pnb272w1</td>
<td>1.2.840.10045.3.0.16</td>
</tr>
<tr>
<td>X9.62 c2pnb304w1</td>
<td>c2pnb304w1</td>
<td>1.2.840.10045.3.0.17</td>
</tr>
<tr>
<td>X9.62 c2tnb359v1</td>
<td>c2tnb359v1</td>
<td>1.2.840.10045.3.0.18</td>
</tr>
<tr>
<td>X9.62 c2pnb368w1</td>
<td>c2pnb368w1</td>
<td>1.2.840.10045.3.0.19</td>
</tr>
<tr>
<td>X9.62 c2tnb431r1</td>
<td>c2tnb431r1</td>
<td>1.2.840.10045.3.0.20</td>
</tr>
<tr>
<td>X9.62 prime192v1</td>
<td>prime192v1</td>
<td>1.2.840.10045.3.1.1</td>
</tr>
<tr>
<td>X9.62 prime192v2</td>
<td>prime192v2</td>
<td>1.2.840.10045.3.1.2</td>
</tr>
<tr>
<td>X9.62 prime192v3</td>
<td>prime192v3</td>
<td>1.2.840.10045.3.1.3</td>
</tr>
<tr>
<td>X9.62 prime239v1</td>
<td>prime239v1</td>
<td>1.2.840.10045.3.1.4</td>
</tr>
<tr>
<td>X9.62 prime239v2</td>
<td>prime239v2</td>
<td>1.2.840.10045.3.1.5</td>
</tr>
<tr>
<td>X9.62 prime239v3</td>
<td>prime239v3</td>
<td>1.2.840.10045.3.1.6</td>
</tr>
<tr>
<td>X9.62 prime256v1</td>
<td>prime256v1</td>
<td>1.2.840.10045.3.1.7</td>
</tr>
</tbody>
</table>
Appendix D – Use with OSGI

In order to properly support the JSSE it was necessary to call some specific sun classes. For this reason many OSGI containers will not be able to immediately resolve the bc-fips jar. If this happens adding:

sun.security.provider
sun.security.internal.spec

to the

org.osgi.framework.system.packages.extra

configuration property for the container should deal with the issue.

Likewise with containers like JBoss it is also necessary to include the paths for “sun/security/provider” and “sun/security/internal/spec” in the module dependencies.
Appendix E – Public/Private Key Conversion

All public/private keys in both the JCE/JCA and the low-level API return the appropriate ASN.1 encoding via their getEncoded() methods.

So, for example, the following shows how to convert a JCA RSA key pair into low level keys:

```java
KeyPairGenerator kpGen = KeyPairGenerator.getInstance("RSA", "BCFIPS");
kpGen.initialize(2048);

KeyPair kp = kpGen.generateKeyPair();

AsymmetricRSAPublicKey rsaPubKey =
    new AsymmetricRSAPublicKey(FipsRSA.Algorithm, kp.getPublic().getEncoded());

AsymmetricRSAPrivateKey rsaPrivKey =
    new AsymmetricRSAPrivateKey(FipsRSA.Algorithm, kp.getPrivate().getEncoded());
```

Going back the other way requires the use of the JCA KeyFactory class. In this case we use the getEncoded() method on the the two low-level keys, but in this case create the corresponding key specs which we then pass to the KeyFactory. For example:

```java
KeyFactory keyFactory = KeyFactory.getInstance("RSA", "BCFIPS");

RSAPublicKey jcaPubKey = (RSAPublicKey)keyFactory.generatePublic(
    new X509EncodedKeySpec(rsaPubKey.getEncoded()));

RSAPrivateKey jcaPrivKey = (RSAPrivateKey)keyFactory.generatePrivate(
    new PKCS8EncodedKeySpec(rsaPrivKey.getEncoded()));
```

will convert the two low level keys created in the first code snippet back into their JCA counterparts.
Appendix F – The BCFKS file Format

The BCFKS KeyStore files are constructed using ASN.1 where the outer layer of the key store is described as follows:

ObjectStore ::= SEQUENCE {
    CHOICE {
        encryptedObjectStoreData EncryptedObjectStoreData,
        objectStoreData ObjectStoreData
    }
    integrityCheck ObjectStoreIntegrityCheck
}

In the case of the BCFKS store the CHOICE item is EncryptedObjectStoreData and the integrity check is HMAC SHA-512 using a 512 bit key derived from the KeyStore password. The algorithm used to derive the key is PKCS#5 Scheme 2 and the password is converted into bytes for the key deriver using the PKCS#12 scheme, so each Java character is converted into 2 bytes. A differentiator is also concatenated with the password so that integrity keys will not equal encryption keys.

EncryptedObjectStoreData is defined as:

EncryptedObjectStoreData ::= SEQUENCE {
    encryptionAlgorithm AlgorithmIdentifier,
    encryptedContent OCTET STRING
}

In this case the algorithm used is 256 bit AES CCM. The outer key is again derived from the password using PKCS#5 scheme2 but uses a different salt as well as a differentiator specific for store encryption to avoid key overlap.

The OCTET STRING representing the encrypted content is an encrypted encoding of an ObjectStoreData structure.

ObjectStoreData ::= SEQUENCE {
    version INTEGER,
    dataSalt OCTET STRING,
    integrityAlgorithm AlgorithmIdentifier,
    creationDate GeneralizedTime,
    lastModifiedDate GeneralizedTime,
    objectDataSequence ObjectDataSequence,
    comment UTF8String OPTIONAL
}

with ObjectDataSequence defined as:

ObjectDataSequence ::= SEQUENCE OF ObjectData

and ObjectData defined as:

ObjectData ::= SEQUENCE {
    type INTEGER,
    identifier UTF8String,
    creationDate GeneralizedTime,
    lastModifiedDate GeneralizedTime,
    data OCTET STRING,
In the case of a BCFKS key store, type will be one of CERTIFICATE(0), PRIVATE_KEY(1), SECRET_KEY(2), PROTECTED_PRIVATE_KEY(3), and PROTECTED_SECRET_KEY(4). The last two types allow for some customisation of the encryption used for storing a private key or a secret key via the setKeyEntry() method that takes a byte array.

Certificates are stored using an encoding of the standard PKIX Certificate type.

In the case of private keys and secret keys, they are stored using either an encoding of an EncryptedPrivateKeyData object or an EncryptedSecretKeyData object.

EncryptedPrivateKeyData is defined as:

```
EncryptedPrivateKeyObjectData ::= SEQUENCE {
  encryptedPrivateKeyInfo EncryptedPrivateKeyInfo,
  certificates SEQUENCE OF Certificate
}
```

where the `encryptedPrivateKeyInfo` is generated using 256 bit AES CCM with a key derived from the password to store the key using the same scheme as with the store itself. A differentiator is also concatenated with the password to avoid key overlap in case the same password is used for a different purpose elsewhere in the KeyStore.

EncryptedSecretKeyData is defined as:

```
EncryptedSecretKeyData ::= SEQUENCE {
  keyEncryptionAlgorithm AlgorithmIdentifier,
  encryptedKeyData OCTET STRING
}
```

where the encrypted key data is an encrypted encoding of a SecretKeyData object defined as:

```
SecretKeyData ::= SEQUENCE {
  keyAlgorithm OBJECT IDENTIFIER,
  keyBytes OCTET STRING
}
```

The encoding of the SecretKeyData is also encrypted using 256 bit AES CCM using a key derived from the password to store the key using the same scheme as with the store itself. A differentiator is also concatenated with the password to avoid key overlap in case the same password is used for a different purpose elsewhere in the KeyStore.
Appendix G – Troubleshooting

1. Use of the BCFIPS provider results in exceptions reporting as:
"java.lang.SecurityException: Unsupported keysize or algorithm parameters"
or "java.security.InvalidKeyException: Illegal key size".

If you see one of these exceptions it means the unrestricted policy files for the JVM you are using have not been installed. For a standard JVM you can navigate to the policy files by going to http://www.oracle.com/technetwork/java/index.html and then looking in the download area for your the JDK/JRE you are using (the links to the policy files are normally at the bottom of the page). Download the zip file provided, follow the instructions, making sure you are installing the files into the JVM you are running with, and you should find the exception stops happening.

Note: Providing maximum key sizes are not exceeded it is possible to use the BCFIPS provider without getting this exception, so a lack of these exceptions in early operation may simply mean nothing has happened to trigger them. If it suddenly starts happening the first thing to check is the policy files.

2. JVM using the BCFIPS provider periodically blocks, especially on start up.

The BCFIPS provider is very careful about making sure it has access to sufficient entropy to generate good quality keys. Occasionally the source the JVM is using will run out of entropy and block. The best way to deal with this is to make sure the execution environment has hardware RNG turned on if it is available.

In the case where this is not possible you can either implement an entropy pool based on a FIPS DRBG and use that to seed your DRBG, or in the case where you are using the JCA/JCE you can configure an entropy pool for the BCFIPS provider by using the HYBRID configuration option (see Chapter 2, “Installation”, Section 2.3).

3. JSSE – cannot find JKS key store type or key store not from BCFIPS provider.

If using the JSSE in FIPS mode, the key stores containing either the private server credentials, or the private client credentials, must be readable using the BCFIPS provider. The only key store type the BCFIPS provider has available that is FIPS compliant is the BCFKS key store type, so when using the JSSE in FIPS mode, the key stores for private key credentials need to be of the type BCFKS, and any container utilising the JSSE needs to be configured appropriately.

For example, in tomcat, the Connector config for SSL/TLS will usually need to include:

```
keystoreType="BCFKS"
keystoreProvider="BCFIPS"
```

To make sure the tomcat passes in the correct key store type to allow the JSSE to work in FIPS mode with the BCFIPS provider.
Appendix H – Disclosures

Please note that just as patents can vary from jurisdiction to jurisdiction in any field of endeavour, the same applies to cryptographic algorithms where some techniques which are freely usable in some places are patented in others. It is your responsibility to make sure you understand the situation with patents as it applies to your circumstances.

In the case of one standard we implement, derived from RFC 5753, we are aware of a disclosure requirement. This is to say, the module contains implementations of EC MQV as described in RFC 5753, “Use of ECC Algorithms in CMS”. In line with the conditions in:


We state, where EC MQV has not otherwise been disabled:

“The use of this product or service is subject to the reasonable, non-discriminatory terms in the Intellectual Property Rights (IPR) Disclosures of Certicom Corp. at the IETF for Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS) implemented in the product or service.”
Appendix I – References
