Frequently Asked Questions

1. Why do I get "`java.lang.SecurityException: Unsupported keysize or algorithm parameters`" or "`java.security.InvalidKeyException: Illegal key size`" when I try using the Bouncy Castle Provider?

If you see this it means the unrestricted policy files for the JVM you are using have not been installed. You can find these at http://www.oracle.com/technetwork/java/index.html at the same place as you downloaded the JDK/JRE (they're normally at the bottom of the page). Download the zip file Sun provide, 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 BC provider without getting this exception. If it suddenly starts happening the first thing to check is the policy files.

2. Where can I find examples of how to use the APIs?

There are specific example programs for dealing with Attribute Certificates, PKCS12, SMIME and OpenPGP. They can be found in the packages:

- org.bouncycastle.jce.examples
- org.bouncycastle.mail.smime.examples
- org.bouncycastle.openpgp.examples

Another useful source of examples is the test packages:

- org.bouncycastle.crypto.test
- org.bouncycastle.jce.provider.test
- org.bouncycastle.cms.test
- org.bouncycastle.mail.smime.test
- org.bouncycastle.openpgp.test
- org.bouncycastle.cert.test
- org.bouncycastle.pkcs.test
- org.bouncycastle.tsp.test

There are also code examples from Beginning Cryptography with Java which demonstrate both the use of the JCE/JCA and also some of the Bouncy Castle APIs.

Finally, there are also links to books and articles on our resources page and further articles elsewhere on this wiki.

3. I am using the lightweight library to create some MIDlets and my device/simulator is complaining about creation of classes in the Java

package (such as java/math/BigInteger, java/security/SecureRandom, java/io/FilterInputStream), don't you Bouncy Castle guys know that's not allowed?

The lightweight library contains some compatibility classes in the java/* namespace to make development easier for compatibility between client/server code (otherwise the MIDlet would be org/bc/math/BigInteger and the Servlet would be java/math/BigInteger) as well as keeping the BC codebase as small as possible. This change was introduced a number of years ago, and announced on the BC mailing list. Since then, many users have been creating MIDlets using the cldc_classes.zip and the world is a happy place.

There is one, fundamental, important step that is required when creating a MIDlet. That is you must, must, must obfuscate the classes. If you do this correctly, everything works fine. If you do not do this correctly, your device/simulator will complain. Correctly in this case means that the package names also need obfuscating, not just the class names. The options for doing this are obfuscator dependent.

Here is a thread from a Nokia forum discussion how to do this in Netbeans.

4. When I encrypt something with RSA I am losing leading zero bytes off my data, why are you guys shipping such a broken implementation?

RSA is not a regular block cipher, its operation is based on big integer arithmetic. As this is the case leading zero bytes will be lost since they are not meaningful (numerically speaking). You can get around this problem with RSA by using one of the padding mechanisms such as PKCS1 or OAEP. To do this in code with the BC provider you need to create the cipher as:

```
Cipher rsaCipher = Cipher.getInstance("RSA/NONE/PKCS1Padding", "BC");
```

or

```
Cipher rsaCipher = Cipher.getInstance("RSA/NONE/OAEPWithSHA1AndMGF1Padding", "BC");
```

If you create the cipher using:

```
Cipher rsaCipher = Cipher.getInstance("RSA", "BC");
```

it is actually up to the provider to decide what sort of padding you get. In the case of the BC provider this is none at all, equivalent to:

```
Cipher rsaCipher = Cipher.getInstance("RSA/NONE/NoPadding", "BC");
```

5. I am trying to encrypt a megabyte of data using a 1024 bit RSA key but when I do I get an error indicating RSA will not process more than 127 bytes of data. Why won't it work?

The RSA implementation that ships with Bouncy Castle only allows the encrypting of a single block of data. The RSA algorithm is not suited to streaming data and should not be used that way. In a situation like this you should encrypt the data using a randomly generated key and a symmetric cipher, after that you should encrypt the randomly generated key using RSA using an appropriate padding algorithm, and then send the encrypted data and the encrypted random key to the other end where they can reverse the process (i.e. decrypt the random key using their RSA private key and then decrypt the data).

6. I am using the DES/DES-EDE encryption algorithm and I've discovered that I can generate a DES/DES-EDE key and change some

of the bits in the key and have the key still be able to decrypt the data, what kind of clowns wrote this implementation? Why can I change some of the key bits and still decrypt?

DES keys actually include parity bits, one per byte, so a 64 bit DES key is in reality only a 56 bit one. The parity bits are ignored by the algorithm when it processes the key, so while you might decide whether a key is valid or not according to the parity bits, the value of the bits set aside for parity checking are not taken into account when initialising the cipher.

7. I want to build Bouncy Castle, but can't work out how to do it. Can you make it easier for me?

You may want to also ask yourself "why?" - it may not be worth it. If you want to build a provider, you'll need a Sun signing key if you are working with the Sun JCE or what you build will not work. This page contains information for those who are brave enough to tread this ground.

8. The reason I need to build is so I can use pack200 to shrink the jar. Is there another way of doing it?

The BC jars for JDK 1.5 and later are all repacked using pack200 before signing, so they can be subsequently be packed/unpacked and still verify. You will need to specify --segment-limit=-1 (or -S-1) as an argument to pack200, otherwise there is a chance the signature will change.

9. Is Bouncy Castle FIPS-140 certified?

We now have FIPS certified API with a release version of 1.0.0. Work on 1.0.1 has also begun. Access to the unreleased 1.0.1 version is available under an early access program which we offer to people and organisations who either donate 6000 USD, or more, to the project or hold Bouncy Castle support contracts, Bronze level or above, through Crypto Workshop . The early access program also includes the CAVP test harness and other documentation in full source. If you are interested in donating to this effort in general you can donate at our donations page or contact us at office@bouncycastle.org if you would like further details. The 1.0.0 version is available at https://www.bouncycastle.org/fips-java

10. If I am using the Bouncy Castle FIPS APIs is my application also FIPS compliant?

In general the answer to this is yes, providing you are using the API in accordance with the security policy that comes with the FIPS API. It is also possible to get third party reviews done if further assurance is required. Please contact us at Crypto Workshop if you would like further details.

11. What is Bouncy Castle's export classification in the United States of America?

Originally (pre 2017) Bouncy Castle was approved classified under ECCN code 5D002 and approved for export under License Exception TSU. As at this time of writing (June, 2017) the ECCN code for open source software like Bouncy Castle that has been registered (as in reviewed by BIS and released from “EI” and “NS” controls pursuant to §742.15(b) of the EAR), is now 5D992.c If you also need to list algorithms available in the provider and the strengths supported, you can find the information in the specifications.html file provided with the distribution you are using. See The Bureau of Industry and Security website for further details.