ZCOM H3204: Cryptography
Practical 3: The Java Cryptography Architecture (JCA) & Bouncy Castle

Martin Harrigan, martin.harrigan@itcarlow.ie

The Java Cryptography Architecture (JCA) provides a foundation for a range of security services for the Java ecosystem. The foundation is specified using the *provider architecture* by having a set of *service classes* provide a uniform interface to developers when interacting with installed *providers of services* (see Fig. 1). In other words, a Java developer only needs to learn the JCA API, to use any of the libraries that support it.

Bouncy Castle is a cryptography library for the Java and C# programming languages. The library is developed by the Legion of the Bouncy Castle Inc. It provides cryptographic services to the JCA.

The provider architecture enables a separation between what an application developer will normally call, for example methods belonging to `javax.crypto.Cipher`, and what is actually implemented by a provider of cryptographic services, such as Bouncy Castle. For example, a developer can create an instance of the Blowfish cipher as follows:

```
Cipher c = Cipher.getInstance("Blowfish/CBC/PKCS7Padding");
```

But an implementor of the Blowfish cipher service will actually extend the `javax.crypto.CipherSpi` class and it will be this Service Provider Interface (SPI) object that is wrapped inside the Cipher object and provides the actual implementation of the cipher.

The provider architecture enables the deployer of the application to determine which vendor implementation of a particular algorithm is used. For example, to specify the Bouncy Castle implementation of the Blowfish cipher service, the developer can use:

```
Cipher c = Cipher.getInstance("Blowfish/CBC/PKCS7Padding", "BC");
```

Note the `"BC"` parameter: this specifies that the developer wants to use the implementation of the Blowfish cipher provided by the Bouncy Castle library.

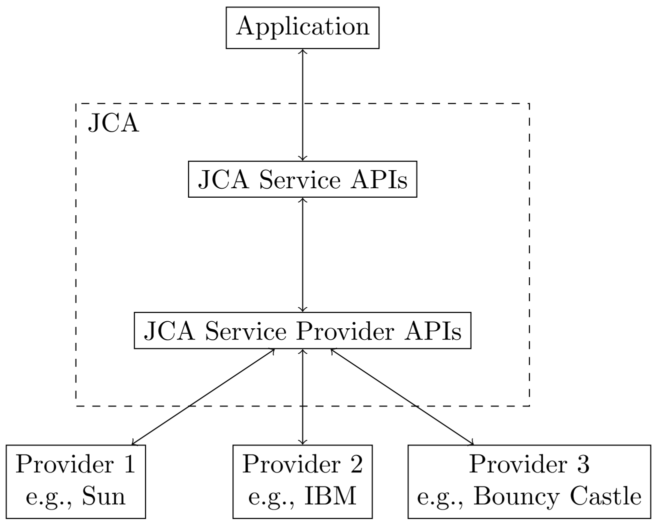

Figure 1: The Java Cryptography Architecture (JCA) separates the application from the provider of cryptographic services. The JCA comprises the JCA Service APIs, which are used by the application, and the JCA Service Provider APIs, which are used by the provider of cryptographic services.

1 What Providers Are Available?

The available providers are accessible using the standard Java API. The following code displays a list of the providers installed in the Java runtime producing their names and 'info' strings as output. Create a Java project that runs the code below and inspect the providers available on your system.

```java
package providers;

import java.security.Provider;
import java.security.Security;

public class ListProviders {

  public static void main(final String[] args) {
    // We get a list of the providers and store them in an array.
    final Provider[] providers = Security.getProviders();

    // We iterate through each provider in the array and print their names
    // and information.
    for (final Provider provider : providers) {
      System.out.print(provider.getName());
      System.out.print(": ");
      System.out.print(provider.getInfo());
      System.out.println();
    }
  }
}
```

Listing 1: ListProviders.java

2 What Capabilities Does a Provider Offer?

The available algorithms offered by a provider can also be accessed. The following code prints out the available algorithms, skipping any aliases. Create a Java project that runs the code below and inspect the capabilities offered by the providers available on your system. You can modify the string `"SunEC"` to select a provider.

```java
package providers;

import java.security.Provider;
import java.security.Security;
import java.util.Iterator;
```

```
public class ListProviderCapabilities {

  public static void main(final String[] args) {
    // We provide the name of a provider we are interested in.
    final Provider provider = Security.getProvider("SunEC");

    // We iterate through everything provided by the provider, e.g., key
    // generators, algorithms or ciphers, etc.
    for (final Iterator<Object> it = provider.keySet().iterator(); it
        .hasNext();) {
      final String entry = (String) it.next();

      if (entry.startsWith("Alg.Alias")) {
        continue; // We can skip "aliases".
      }

      // For each entry we print its type (or "serviceName") and its name
      // ("name").
      final String serviceName = entry.substring(0, entry.indexOf('.'));
      final String name = entry.substring(serviceName.length() + 1);
      System.out.println(serviceName + ": " + name);
    }
  }
}
```

Listing 2: ListProviderCapabilities.java

3 Download and Setup the Bouncy Castle Provider

In this, and future labs, we will download and install the Bouncy Castle provider.

You can download the latest version of Bouncy Castle for Java at `https://www.bouncycastle.org/latest_releases.html`. You need to find the (signed) JAR file for your JDK. At the time of writing, the latest JAR file is called `bcprov-jdk15on-169.jar`. Download this file and add it to the project you created in Listing 1. See `https://www.youtube.com/watch?v=lXOFhmpHGrc` for the steps to add a JAR file to an Eclipse project. This video is based on a MySQL JAR; you should adapt the steps for the Bouncy Castle JAR.

Modify the code as follows and check that Bouncy Castle is listed as a provider.

```
package providers;

import java.security.Provider;
import java.security.Security;
```

```java
import org.bouncycastle.jce.provider.BouncyCastleProvider;

public class ListProvidersWithBC {

  static {
    // We add Bouncy Castle as a provider.
    Security.addProvider(new BouncyCastleProvider());
  }

  public static void main(final String[] args) {
    // We get a list of the providers and store them in an array.
    final Provider[] providers = Security.getProviders();

    // We iterate through each provider in the array and print its name and
    // information. Note the inclusion of the "BC" (Bouncy Castle) provider.
    for (final Provider provider : providers) {
      System.out.print(provider.getName());
      System.out.print(": ");
      System.out.print(provider.getInfo());
      System.out.println();
    }
  }
}
```

Listing 3: ListProvidersWithBC.java

Make similar modifications to the code in Listing 2 to inspect the capabilities of the Bouncy Castle library.

4 Encrypt and Decrypt using AES

AES is an algorithm for the symmetric encryption and decryption of data. The following code uses Bouncy Castle's implementation of AES to encrypt and decrypt a message using a key. Can you measure how long it takes to perform the encryption and decryption on your computer? Hint: You can use `System.currentTimeMillis()` to get the current time in milliseconds.

```java
package aes;

import java.security.InvalidKeyException;
import java.security.Key;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.Security;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.SecretKeySpec;

import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.bouncycastle.util.encoders.Hex;

public class AESEncryptDecrypt {

  static {
    Security.addProvider(new BouncyCastleProvider());
  }

  // The three methods below throw "checked exceptions", e.g.,
  // NoSuchAlgorithmException and NoSuchProviderException. An exception is
  // generated with something exceptional happens, e.g., an algorithm like AES
  // or a provider like Bouncy Castle cannot be found. A "checked exception"
  // is an exception that we must handle in some way. In the code below we
  // simply "throw" the checked exceptions so that the program terminates with
  // a stack trace.

  private static byte[] encrypt(final Cipher cipher, final Key key,
      final byte[] data) throws InvalidKeyException,
      IllegalBlockSizeException, BadPaddingException {
    cipher.init(Cipher.ENCRYPT_MODE, key);
    return cipher.doFinal(data);
  }

  private static byte[] decrypt(final Cipher cipher, final Key key,
      final byte[] data) throws InvalidKeyException,
      IllegalBlockSizeException, BadPaddingException {
    cipher.init(Cipher.DECRYPT_MODE, key);
    return cipher.doFinal(data);
  }

  public static void main(final String[] args)
      throws NoSuchAlgorithmException, NoSuchProviderException,
      NoSuchPaddingException, InvalidKeyException,
      IllegalBlockSizeException, BadPaddingException {
    final String keytext = "thebestsecretkey";
    System.out.println("Key: " + keytext);

    final Cipher cipher = Cipher.getInstance("AES", "BC");
    final Key key = new SecretKeySpec(keytext.getBytes(), "AES");
    final String plaintext = "Hello world!";
    System.out.println("Plaintext: " + plaintext);
    final byte[] ciphertext = encrypt(cipher, key, plaintext.getBytes());

    System.out.println("Ciphertext: " + Hex.toHexString(ciphertext));
    final String plaintext2 = new String(decrypt(cipher, key, ciphertext));

    System.out.println("Plaintext (decrypted): " + plaintext2);

    assert (plaintext.equals(plaintext2));
  }
}
```

Listing 4: AESEncryptDecrypt.java

References

Legion of the Bouncy Castle Inc. The Bouncy Castle Crypto APIs. `https://www.bouncycastle.org`, 2020.