Using the Bouncy Castle Provider's ImplicitlyCA Facility

X9.62 provides 3 alternatives for the parameters that can be found in an EC public key.

One of these is named implicitlyCA and indicates that the parameters are defined else where, implicit in the name of the certification authority (CA) that issued the key. In this stiuation the actual parameters appear in the ASN.1 encoding of the key as a DER encoded NULL, and calling getParameters on BC `ECKey` interface, or `getParams()` on the JDK ECKey interface will return the `null` value.

As the definition says, when the key is used, the parameters will have to come from elsewhere. In the case of the BC provider the interface `ConfigurableProvider` is defined in the `org.bouncycastle.jce.interfaces` package. The interface is implemented by the Bouncy Castle provider and allows for the configuration of implicit parameters where they are required using one of two modes:

- providing the parameters so they can be accessed anywhere within the same JVM using `ConfigurableProvider.EC_IMPLICITLY_CA`
- providing the parameters so they can only be accessed by the current thread using `ConfigurableProvider.THREAD_LOCAL_EC_IMPLICITLY_CA`.

In some cases having unlimited access to this facility in a JVM would not be regarded as a god thing. If necessary the ability to access this facility can also be protected using the `org.bouncycastle.jce.ProviderConfigurationPermission`. Details about this are described in the last section of this document.

Global Setup of ImplicitlyCA Using the Bouncy Castle EC API

We can use the `ConfigurableProvider` interface to set the implicitlyCA parameter as follows:

```
import org.bouncycastle.math.ec.ECCurve;
import org.bouncycastle.jce.spec.ECParameterSpec;
...
ECCurve curve = new ECCurve.Fp(
            new
BigInteger("8834235323891921647916487503603088853144765972529603627924508606096998
39"), // q
            new
BigInteger("7fffffffffffffffffffff7fffffffffff8000000000007ffffffffffc", 16), //
a
            new
BigInteger("6b016c3bdcf18941d0d654921475ca71a9db2fb27d1d37796185c2942c0a", 16)); //
b
ECParameterSpec ecSpec = new ECParameterSpec(
            curve,

curve.decodePoint(Hex.decode("020ffa963cdca8816ccc33b8642bedf905c3d358573d3f27fbbd3
b3cb9aaaf")), // G
            new
BigInteger("8834235323891921647916487503603088848075503416916277522753454247028073
07")); // n
ConfigurableProvider config = (ConfigurableProvider)Security.getProvider("BC");
config.setParameter(ConfigurableProvider.EC_IMPLICITLY_CA, ecSpec);
```

From this point on any key with a null parameters field will be assumed to be using the `ECParameterSpec` that was passed in.

Global Setup of ImplicitlyCA Using the JDK EC (JDK 1.5 or later)

Alternately the implicitlyCA parameter can be set using an `ECParameterSpec` object from the JDK's `java.security.spec` package. The

procedure is the same as that for the BC `ECParameterSpec` and the provider will internally translate the object into the format it needs.

```
import java.security.spec.EllipticCurve;
import java.security.spec.ECFieldFp;
import java.security.spec.ECParameterSpec;
...
EllipticCurve curve = new EllipticCurve(
            new ECFieldFp(new
BigInteger("883423532389192164791648750360308885314476597252960362792450860609699839")), // q
            new
BigInteger("7fffffffffffffffffffff7fffffffffff8000000000007ffffffffc", 16), // a
            new
BigInteger("6b016c3bdcf18941d0d654921475ca71a9db2fb27d1d37796185c2942c0a", 16)); // b
ECParameterSpec ecSpec = new java.security.spec.ECParameterSpec(
            curve,
            ECPointUtil.decodePoint(curve,
Hex.decode("020ffa963cdca8816ccc33b8642bedf905c3d358573d3f27fbbd3b3cb9aaaf")), // G
            new
BigInteger("883423532389192164791648750360308884807550341691627752275345424702807307"), // n
            1); // h

ConfigurableProvider config = (ConfigurableProvider)Security.getProvider("BC");
config.setParameter(ConfigurableProvider.EC_IMPLICITLY_CA, ecSpec);
```

Thread Local Setup of the ImplicitlyCA Parameter

The thread local case is the same as the setup of the global case and can also be used with both the BC API or the JDK API for elliptic curve.

For example, using the JDK 1.4 API you would write something like the following:

```
import org.bouncycastle.math.ec.ECCurve;
import org.bouncycastle.jce.spec.ECParameterSpec;
...
ECCurve curve = new ECCurve.Fp(
            new
BigInteger("883423532389192164791648750360308885314476597252960362792450860609699839"), // q
            new
BigInteger("7fffffffffffffffffffff7fffffffffff8000000000007ffffffffc", 16), // a
            new
BigInteger("6b016c3bdcf18941d0d654921475ca71a9db2fb27d1d37796185c2942c0a", 16)); // b
ECParameterSpec ecSpec = new ECParameterSpec(
            curve,

curve.decodePoint(Hex.decode("020ffa963cdca8816ccc33b8642bedf905c3d358573d3f27fbbd3b3cb9aaaf")), // G
            new
BigInteger("883423532389192164791648750360308884807550341691627752275345424702807307")); // n
ConfigurableProvider config = (ConfigurableProvider)Security.getProvider("BC");
config.setParameter(ConfigurableProvider.THREAD_LOCAL_EC_IMPLICITLY_CA, ecSpec);
```

You can also pass `config.setParameter()` `null` as a parameter value. This can be useful if you not only want to restrict the use of

implicitlyCA to a particular thread, but to a particular section of code as well. For example, given a method called `getImplicitParams()` w hich takes a CA name and returns the CA's implicit parameters you could use the following code to carry out restricted operations:

```
try
{
    config.setParameter(ConfigurableProvider.THREAD_LOCAL_EC_IMPLICITLY_CA,
getImplicitParams("myCA"));
    // EC operations such as use of KeyFactory, KeyGeneration, and Signature
processing go here
}
finally
{
    config.setParameter(ConfigurableProvider.THREAD_LOCAL_EC_IMPLICITLY_CA, null);
}
```

Using the ImplicitlyCA Parameter to Generate Keys

Telling the BC provider to make use of the implicit parameters is then just a matter of passing in a null where a parameter object is expected, for example, to create a key pair:

```
ConfigurableProvider.setParameter(..., ecParams);
...
KeyPairGenerator g = KeyPairGenerator.getInstance("ECDSA", "BC");
g.initialize(null, new SecureRandom());
KeyPair p = g.generateKeyPair();
ECPrivateKey sKey = (ECPrivateKey)p.getPrivate();
ECPublicKey vKey = (ECPublicKey)p.getPublic();
```

The resulting keys, vKey and sKey can then be used to create ECDSA signatures, in the same manner as normal EC keys.

Note: it is important to remember that keys generated in this fashion will be encoded with their parameters block set to ASN.1 NULL. If you need to export them to another installation the other installation must have the same parameters set for ImplictlyCA to be able to use the keys.

Using the ImplicitlyCA Parameter with KeyFactory

In this case nothing is required other than making sure the implictlyCA parameter is set for either the current thread or globally for the VM. Once that is done the {{KeyFactory}} class can be used as normal:

```
ConfigurableProvider.setParameter(..., ecParams);
.....
PKCS8EncodedKeySpec privSpec = new PKCS8EncodedKeySpec(getEncoded_of_private_key);
X509EncodedKeySpec pubSpec = new X509EncodedKeySpec(getEncoded_of_public_key);
KeyFactory keyFact = KeyFactory.getInstance("ECDSA", "BC");
PrivateKey privKey = keyFact.generatePrivate(privSpec);
PublicKey  pubKey = keyFact.generatePublic(pubSpec);
```

Limiting Access to the ImplicitlyCA Facility Using Java Policy

In situations where you need to limit access to the implicitlyCA feature in the Bouncy Castle provider you can restrict it by running the JVM with a security manager and using a policy file.

As a simple example, assuming you are trying to run the implicitlyCA test case and you are including a security manager you would need the following command line:

```
java -Djava.security.manager -Djava.security.policy=test.policy
org.bouncycastle.jce.provider.test.ImplicitlyCaTest
```

and, at a minimum, the following policy file:

```
grant {
   permission java.security.SecurityPermission "putProviderProperty.BC";
   permission java.security.SecurityPermission "insertProvider.BC";
   permission org.bouncycastle.jce.ProviderConfigurationPermission "BC",
"ecImplicitlyCA, threadLocalEcImplicitlyCA";
};
```

After that restricting the use of ImplicitlyCA is a matter of adding further restriction to the `grant` term in the policy statement. Note that either of the global or thread local implictlyCA facilities can be disabled by leaving out the corresponding action keyword in the `ProviderConfigurationPermission` declaration.