SCAPI Project

[1. Introduction 1](#_Toc308690244)

[2. Low-level crypto functions 1](#_Toc308690245)

[3. External Libraries 2](#_Toc308690246)

[3.1 Java 2](#_Toc308690247)

[3.2 C/C++ 2](#_Toc308690248)

[4. Usage 2](#_Toc308690249)

[4.1 Factories 2](#_Toc308690250)

[5. Dlog group description 3](#_Toc308690251)

[5.1 Dlog concrete implementations 5](#_Toc308690261)

[5.2 Dlog group usage examples 7](#_Toc308690262)

[6. Curves file format 10](#_Toc308690263)

[7. SCAPI Installation 12](#_Toc308690264)

# Introduction

SCAPI is a service for high-level multi-party computation protocols as well as a standalone product. High-level protocols will be built upon the low-level primitives in SCAPI. SCAPI is composed of the following three layers.

1. **Low-level crypto functions:** these are functions that are basic building blocks for cryptographic constructions (e.g., pseudorandom functions and pseudorandom generators belong to this layer).
2. **Non-interactive mid-level crypto functions:** these are non-interactive functions that can be applications within themselves in addition to being tools (e.g., encryption and signature schemes belong to this layer).
3. **Mid-level interactive crypto protocols:** these are interactive protocols involving two or more parties; typically, the protocols in this layer are popular building blocks like commitments, zero knowledge and oblivious transfer, but being a popular building block is not a requirement.

# Low-level crypto functions

This layer includes the following cryptographic primitives:

* Pseudorandom generators
* Pseudorandom functions
* Pseudorandom permutations
* Collision resistant hash functions
* Universal hash functions
* Trapdoor permutations
* Get (true) random
* A discrete log library
* Key derivation functions

However, since we are still in the middle of the developing process, we are currently providing you only with the DLog Group capabilities.

# External Libraries

The infrastructure is built upon several cryptographic packages. The packages are divided into two main groups, libraries that are written in Java and libraries in native languages such as C/C++.

### Java

* + Java JDK and Sun's different Providers
  + Bouncy castle

Bouncy Castle uses the provider architecture of the security package of Java and it has interfaces that are strongly related to our required interfaces thus, is easy to integrate into this layer. In order to use the BC code we will write wrappers for written classes to match our primitive layer API interfaces.

### C/C++

* + Crypto++ (C++)
  + Miracl (C)

An interface dll translating from Crypto++ and Miracl to Java will be provided using the JNI framework. JNI is a programming platform that allows Java code running in a JVM to call and to be called by native applications and libraries written in other languages, such as C and C++. On top of these dll interfaces, a wrapper will be written to match our java interfaces mentioned above.

# Usage

Elements in SCAPI can be instantiated in two different ways, using SCAPI as a provider or using a factory mechanism. For the moment only the factory mechanism is available.

## Factories

Different elements in this layer can be created using relevant factories. For each family of cryptographic primitives there exists a factory that returns objects of that type.

This is the preferred way to create SCAPI objects if the user wants to rely on SCAPI’s architecture and extensive library of cryptographic tools. SCAPI presents a cryptographic approach to these tools. For example, we use RC4 as a PRG in the lowest layer as well as a stream-cipher in higher layers. We only guarantee support in future versions for elements returned by the Factories.

A list of the factories follows:

* PrfFactory – Factory for pseudorandom functions
* PrgFactory – Factory for pseudorandom generators
* CollResFactory – Factory for collision resistant functions
* PerfectUniversalFactory- Factory for perfect universal functions
* TrapdoorPermutationFactory – Factory for trapdoor permutations
* DlogGroupFactory – Factory for Dlog groups
* KdfFactory – Factory for key derivation functions

Note: In this version you can only see the DlogGroupFactory.

Each such factory has two “getObject” methods. One returns a default implementation (chosen by SCAPI) for the requested element and another one allows the caller to specify the implementation required for the element such as BC, crypto++ etc.

In the sample code below, the caller relies on SCAPI’s choice for the underlying implementation of the AES algorithm. It could be the Bouncy Castle implementation, Crypto++’s implementation or any other.

//Create key, in, out byte arrays

…

//instantiate the prp

PseudorandomPermutation prp = prfFactory.getInstance().getObject("AES");

//init the prp

prp.init(secretKey);

//compute the function with input in and output out.

prp.computeBlock(in, 0, out, 0);

# Dlog group description

We will focus now on a general description of the Dlog Group implementation and its usage.



The **discrete logarithm problem** is as follows: given a generator *g* of a finite group *G* and a random element *h*  *G*, find the (unique) integer *x* such that *gx* = *h*. In cryptography, we are interested in groups for which the discrete logarithm problem (Dlog for short) is assumed to be hard. In fact, although we refer to a “Dlog group”, we typically consider groups in which the decisional Diffie-Hellman (DDH) problem is hard (meaning that *gx,gy,gz* is indistinguishable from *gx,gy,gxy* for random *x,y,z*). The two most common classes are the group **Z***p*\* for a large *p*, and some Elliptic curve groups.

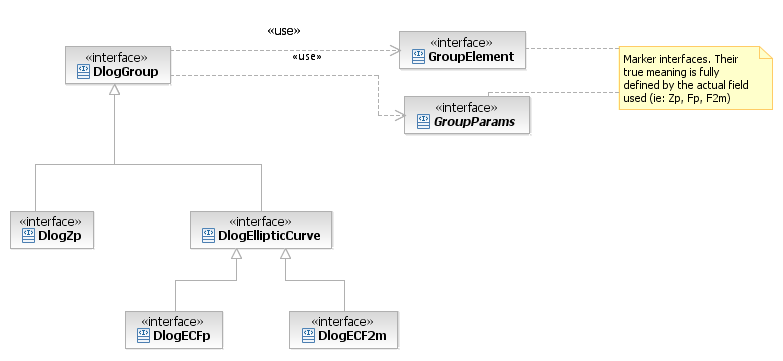
We provide the implementation of the most important Dlog groups in cryptography (see diagram below):

* Elliptic curve over the field
* Elliptic curve over the field

Although Elliptic curves groups look very different from Zp groups, the discrete log problem over EC can be described as follows. Given an elliptic curve E over a finite field F, a base point on that curve P (i.e., a generator of the group defined by the curve), and a random point Q on the curve, the problem is to find the integer n such that nP=Q.

We will use the elliptic curves recommended by NIST.

In the following class diagram, we present the API for the implementation of the different Dlog groups.



Each one of the Dlog groups we implement, is represented by an interface. The hierarchy of the Dlog group, presented in the diagram above allows us to refer either to a Dlog group in general or to a specific one if needed. This is achieved by introducing all the functionality a Dlog group needs in the root interface, named DlogGroup. If a protocol or algorithm needs a specific DlogGroup and not a generic one it can refer specifically to it by the coresponding specific interface: DlogZp, DlogECFp or DlogECF2m. This way the API remains generic and flexible.

## Dlog concrete implementations

As we mentioned above, we provide three types of Dlog groups:

* Elliptic curve over the field
* Elliptic curve over the field

There are various libraries that implement these groups. We incorporate the following libraries implementations:

* Crypto++ implementation of
* Bouncy castle implementation over elliptic curves
* MIRACL implementation over elliptic curves. MIRACL’s performance is shown in Reference [6] - <http://ite.gmu.edu/~kgaj/publications/conferences/GMU_SPEED_2007.pdf>.

There is a concrete class for every library implementation of every DlogGroup.

The initialization of the concrete classes is different and depends on the implementation and the group.

* To initialize a Dlog group over Zp we need to call its init function with an instance of ZpGroupParams.
* Elliptic curve gets the necessary parameters from files with a strict format described [below](#_Curves_file_format).

All the Dlog groups have the following functionality:

|  |  |
| --- | --- |
| [GroupElement](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\GroupElement.html) | [**exponentiate**](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\DlogGroup.html#exponentiate(java.math.BigInteger, edu.biu.scapi.primitives.dlog.GroupElement))(java.math.BigInteger exponent, [GroupElement](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\GroupElement.html) base) Raises the base GroupElement to the exponent. |
| [GroupElement](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\GroupElement.html) | [**exponentiateWithPreComputedValues**](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\DlogGroup.html#exponentiateWithPreComputedValues(edu.biu.scapi.primitives.dlog.GroupElement, int))([GroupElement](file:///C:\\work\\LAST_Project\\SDK\\Code\\doc_only_Dlog2\\edu\\biu\\scapi\\primitives\\dlog\\GroupElement.html" \o "interface in edu.biu.scapi.primitives.dlog) groupElement, int exponent) Computes the product of several exponentiations of the same base and distinct exponents. |
| [GroupElement](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\GroupElement.html) | [**getGenerator**](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\DlogGroup.html#getGenerator())() The generator g of the group is an element of the group such that, when written multiplicatively, every element of the group is a power of g. |
| [GroupParams](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\groupParams\GroupParams.html) | [**getGroupParams**](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\DlogGroup.html#getGroupParams())() GroupParams is a structure that holds the actual data that makes this group a specific Dlog group. |
| java.lang.String | [**getGroupType**](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\DlogGroup.html#getGroupType())() Each concrete class implementing this interface returns a string with a meaningful name for this type of Dlog group. |
| [GroupElement](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\GroupElement.html) | [**getInverse**](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\DlogGroup.html#getInverse(edu.biu.scapi.primitives.dlog.GroupElement))([GroupElement](file:///C:\\work\\LAST_Project\\SDK\\Code\\doc_only_Dlog2\\edu\\biu\\scapi\\primitives\\dlog\\GroupElement.html" \o "interface in edu.biu.scapi.primitives.dlog) groupElement) Calculates the inverse of the given GroupElement. |
| java.math.BigInteger | [**getOrder**](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\DlogGroup.html#getOrder())() |
| [GroupElement](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\GroupElement.html) | [**getRandomElement**](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\DlogGroup.html#getRandomElement())() Creates a random member of this Dlog group |
| boolean | [**isGenerator**](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\DlogGroup.html#isGenerator())() Checks if the element set as the generator is indeed the generator of this group. |
| boolean | [**isInitialized**](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\DlogGroup.html#isInitialized())() Checks if this DlogGroup object has been previously initialized. |
| boolean | [**isMember**](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\DlogGroup.html#isMember(edu.biu.scapi.primitives.dlog.GroupElement))([GroupElement](file:///C:\\work\\LAST_Project\\SDK\\Code\\doc_only_Dlog2\\edu\\biu\\scapi\\primitives\\dlog\\GroupElement.html" \o "interface in edu.biu.scapi.primitives.dlog) element) Checks if the given element is a member of this Dlog group |
| boolean | [**isOrderGreaterThan**](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\DlogGroup.html#isOrderGreaterThan(int))(int numBits) Checks if the order of this group is greater than 2^numBits |
| boolean | [**isPrimeOrder**](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\DlogGroup.html#isPrimeOrder())() Checks if the order is a prime number |
| [GroupElement](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\GroupElement.html) | [**multiplyGroupElements**](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\DlogGroup.html#multiplyGroupElements(edu.biu.scapi.primitives.dlog.GroupElement, edu.biu.scapi.primitives.dlog.GroupElement))([GroupElement](file:///C:\\work\\LAST_Project\\SDK\\Code\\doc_only_Dlog2\\edu\\biu\\scapi\\primitives\\dlog\\GroupElement.html" \o "interface in edu.biu.scapi.primitives.dlog) groupElement1, [GroupElement](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\GroupElement.html) groupElement2) Multiplies two GroupElements |
| [GroupElement](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\GroupElement.html) | [**simultaneousMultipleExponentiations**](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\DlogGroup.html#simultaneousMultipleExponentiations(edu.biu.scapi.primitives.dlog.GroupElement[], java.math.BigInteger[]))([GroupElement](file:///C:\\work\\LAST_Project\\SDK\\Code\\doc_only_Dlog2\\edu\\biu\\scapi\\primitives\\dlog\\GroupElement.html" \o "interface in edu.biu.scapi.primitives.dlog)[] groupElements, java.math.BigInteger[] exponentiations) Computes the product of several exponentiations with distinct bases and distinct exponents. |
| boolean | [**validateGroup**](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog2\edu\biu\scapi\primitives\dlog\DlogGroup.html#validateGroup())() Checks parameters of this group to see if they conform to the type this group is supposed to be. |

All the Elliptic curve Dlog groups implement:

|  |  |
| --- | --- |
| [ECElement](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog\edu\biu\scapi\primitives\dlog\ECElement.html) | [**getElement**](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog\edu\biu\scapi\primitives\dlog\DlogEllipticCurve.html#getElement(java.math.BigInteger, java.math.BigInteger))(java.math.BigInteger x, java.math.BigInteger y) Creates a point with the given x,y values |
| void | [**init**](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog\edu\biu\scapi\primitives\dlog\DlogEllipticCurve.html#init(java.lang.String))(java.lang.String nistCurveName) Initializes the DlogGroup with one of NIST recommended elliptic curve |
| void | [**init**](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog\edu\biu\scapi\primitives\dlog\DlogEllipticCurve.html#init(java.lang.String, java.lang.String))(java.lang.String fileName, java.lang.String curveName) Initializes the DlogGroup with elliptic curve other than NIST curves |

Dlog group over Zp\* implements:

|  |  |
| --- | --- |
| [ZpElement](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog\edu\biu\scapi\primitives\dlog\ZpElement.html) | [**getElement**](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog\edu\biu\scapi\primitives\dlog\DlogZp.html#getElement(java.math.BigInteger))(java.math.BigInteger x) Creates an element with the given x value |
| void | [**init**](file:///C:\work\LAST_Project\SDK\Code\doc_only_Dlog\edu\biu\scapi\primitives\dlog\DlogZp.html#init(edu.biu.scapi.primitives.dlog.groupParams.ZpGroupParams))([ZpGroupParams](file:///C:\\work\\LAST_Project\\SDK\\Code\\doc_only_Dlog\\edu\\biu\\scapi\\primitives\\dlog\\groupParams\\ZpGroupParams.html" \o "class in edu.biu.scapi.primitives.dlog.groupParams) groupParams) Initialize the CryptoPP implementation of Dlog over Zp\* |

## Dlog group usage examples

1. *For elliptic curves over Zp with default provider:*

DlogEllipticCurve myDG = DlogGroupFactory.getInstance().getObject("DlogECFp");

myDG.init("P-192");

//Get a random element

GroupElement el1 = myDG.getRandomElement();

//Get a specific element from some source. An element of a elliptic curve is a point in the //curve with coordinates (x,y) of type BigInteger

GroupElement el2 = myDG.getElement(x, y);

GroupElement el3 = myDG. [multiplyGroupElements](file:///C:\work\LAST_Project\SDK\Code\doc_09_10_11\edu\biu\scapi\primitives\dlog\DlogGroup.html#multiplyGroupElements(edu.biu.scapi.primitives.dlog.GroupElement, edu.biu.scapi.primitives.dlog.GroupElement))(el1,el2);

1. *For elliptic curves over F2m with default provider:*

DlogEllipticCurve myDG = DlogGroupFactory.getInstance().getObject("DlogECF2m");

myDG.init("B-163");

GroupElement el1 = myDG.getRandomElement();

GroupElement el2 = myDG.getRandomElement();

GroupElement el3 = myDG. [multiplyGroupElements](file:///C:\work\LAST_Project\SDK\Code\doc_09_10_11\edu\biu\scapi\primitives\dlog\DlogGroup.html#multiplyGroupElements(edu.biu.scapi.primitives.dlog.GroupElement, edu.biu.scapi.primitives.dlog.GroupElement))(el1,el2);

1. *For Dlog group over Zp with default provider:*

DlogZp myDG = DlogGroupFactory.getInstance().getObject("DlogZp");

myDG.init( new ZpGroupParams(q, g, p));

//Get a random element

GroupElement el1 = myDG.getRandomElement();

//Get a specific element x from some source

BigInteger x = …

GroupElement el2 = myDG.getElement(x);

GroupElement el3 = myDG. [multiplyGroupElements](file:///C:\work\LAST_Project\SDK\Code\doc_09_10_11\edu\biu\scapi\primitives\dlog\DlogGroup.html#multiplyGroupElements(edu.biu.scapi.primitives.dlog.GroupElement, edu.biu.scapi.primitives.dlog.GroupElement))(el1,el2);

1. *For Dlog group over Zp with specific provider:*

//Obtain an instance of a Dlog group over Zp from the Crypto++ provider's implementation.

DlogZpSafePrime myDG = (DlogZpSafePrime) DlogGroupFactory.getInstance().getObject("DlogZpSafePrime", "BC");

BigInteger p = new BigInteger("146962825862718166776543021788972966180494071208157097829415032173221678937321582502321702558186086653661566008474254980895512975381072719365373704524394962496308607917415688320356015815165115812583766234562154742139350468850514297868710965718607352480402620915332262361641848926581798576205267615755040869887");

BigInteger q = new BigInteger("73481412931359083388271510894486483090247035604078548914707516086610839468660791251160851279093043326830783004237127490447756487690536359682686852262197481248154303958707844160178007907582557906291883117281077371069675234425257148934355482859303676240201310457666131180820924463290899288102633807877520434943");

BigInteger g = new BigInteger("2");

myDG.init( new ZpGroupParams(q, g, p ));

//Get a random element

ZpSafePrimeElementCryptoPp el1 =(ZpSafePrimeElementCryptoPp) myDG.getRandomElement();

System.out.println("first random element: " + el1.getElementValue());

ZpSafePrimeElementCryptoPp el2 = (ZpSafePrimeElementCryptoPp)myDG.getRandomElement();

System.out.println("second random element: " + el2.getElementValue());

ZpSafePrimeElementCryptoPp el3 = (ZpSafePrimeElementCryptoPp)myDG. multiplyGroupElements(el1,el2);

System.out.println("The result of multiplication is: " + el3.getElementValue());

If an implementation from a specific provider is required then the getObject(String dlogGroupName, String providerName) has to be used as shown in 4. The following table shows what code name to use for each library.

|  |  |
| --- | --- |
| **Library** | **Code name** |
| Bouncy Castle | BC |
| Miracl | Miracl |
| Crypto++ | CryptoPP |

The table below shows what Dlog groups implementations are taken from which library:

|  |  |  |
| --- | --- | --- |
| **Library** | **Dlog group** | **Language** |
| Bouncy Castle | Elliptic curve over Zp | Java |
| Bouncy Castle | Elliptic curve over F2m | Java |
| Miracl | Elliptic curve over Zp | C |
| Miracl | Elliptic curve over F2m | C |
| Crypto++ | Dlog group over Zp\* | C++ |

The table below shows the list of the names of the NIST elliptic curves that have to be used to initialize the curve as shown in the example above. (myDG.init("P-192"); or myDG.init("B-163"); *)*

|  |
| --- |
| **Curve name** |
| P-192 |
| P-224 |
| P-256 |
| P-384 |
| P-521 |
| B-163 |
| K-163 |
| B-233 |
| K-233 |
| B-283 |
| K-283 |
| B-409 |
| K-409 |
| B-571 |
| K-571 |

The table below shows the list of the names of specific groups to be passed as arguments to the getObject(String algName…) functions of the DlogGroupFactory. The Dlog group name has to be passed as a String exactly as shown in the table.

|  |  |
| --- | --- |
| **Dlog group type** | **Dlog group name** |
| Elliptic curve over Zp | DlogECFp |
| Elliptic curve over F2m | DlogECF2m |
| Dlog group over Zp\* with p a safe prime | DlogZpSafePrime |

# Curves file format

As a default, we will use NIST recommended elliptic curves. We supply in our library a NIST file containing the parameters for the NIST most recommended curves. The concrete EC classes have an init function that accepts only one string representing NIST’s curve’s name, such as P-256. These init functions check that the accepted curve is one of NIST curves and extract the matching parameters from the NIST file. We assume that NIST curves will be used commonly, so this file is uploaded once by the DlogGroupEC class at the start of the program.

Alternatively, the user may choose to use different curves other that NIST recommended curves. To do so, the user needs to create a file and write there the parameters of the curve he wants to use. The format of the files can be found below.

It is the user's responsibility to check the validity of the parameters. We assume that they are valid and do not check them. After the file has been created, the user can call the init function, which accepts two strings that represent the file name and the curve name, respectively. This function opens the file with the given name, extracts the parameters that match the given curve name and sets the generator and the GroupParams of the DlogGroup.

Below there is an example of the format of the curves properties file. This example is taken from NIST curves file, which is attached to our library. A user who wants to use curves other than NIST recommended ones should provide a file in the following format:

# In all the curves, the field elements b, x and y are given in hex and the other #variables are given in

# decimal form. Note that an Elliptic curve group is defined by the prime *p* over   
#which we work, the values *a* and *b* defining the curve *y2=x3+ax+b*, the order *r* of the #group defined by the group, and the base point *G* defined by the pair (G*x,Gy*).

# P-192 is the curve name. This is a **pseudo random** curve over the **Fp** field.

P-192 = 6277101735386680763835789423207666416083908700390324961279

P-192a = -3

P-192b = 64210519 e59c80e7 0fa7e9ab 72243049 feb8deec c146b9b1

P-192r = 6277101735386680763835789423176059013767194773182842284081

P-192x = 188da80e b03090f6 7cbf20eb 43a18800 f4ff0afd 82ff1012

P-192y = 07192b95 ffc8da78 631011ed 6b24cdd5 73f977a1 1e794811

# B-163 is the curve name. This is a **pseudo random** curve over the F2m field with **pentanomial** basis.

B-163 = 163

B-163k = 7

B-163k2 = 6

B-163k3 = 3

B-163a = 1

B-163b = 00000002 0a601907 b8c953ca 1481eb10 512f7874 4a3205fd

B-163r = 5846006549323611672814742442876390689256843201587

B-163x = 00000003 f0eba162 86a2d57e a0991168 d4994637 e8343e36

B-163y = 00000000 d51fbc6c 71a0094f a2cdd545 b11c5c0c 797324f1

# This is a **koblitz** curve over the F2m field with **pentanomial** basis.

K-163 = 163

K-163k = 7

K-163k2 = 6

K-163k3 = 3

K-163a = 1

K-163b = 01

K-163r = 5846006549323611672814741753598448348329118574063

K-163x = 00000002 fe13c053 7bbc11ac aa07d793 de4e6d5e 5c94eee8

K-163y = 00000002 89070fb0 5d38ff58 321f2e80 0536d538 ccdaa3d9

# B-233 is the curve name. This is a **pseudo random** curve over the F2m field with **trinomial** basis.

B-233 = 233

B-233k = 74

B-233a = 1

B-233b = 00000066 647ede6c 332c7f8c 0923bb58 213b333b 20e9ce42 81fe115f 7d8f90ad

B-233r = 6901746346790563787434755862277025555839812737345013555379383634485463

B-233x = 000000fa c9dfcbac 8313bb21 39f1bb75 5fef65bc 391f8b36 f8f8eb73 71fd558b

B-233y = 00000100 6a08a419 03350678 e58528be bf8a0bef f867a7ca 36716f7e 01f81052

# This is a **koblitz** curve over the F2m field with **trinomial** basis.

K-233 = 233

K-233k = 74

K-233a = 0

K-233b = 01

K-233r = 3450873173395281893717377931138512760570940988862252126328087024741343

K-233x = 00000172 32ba853a 7e731af1 29f22ff4 149563a4 19c26bf5 0a4c9d6e efad6126

K-233y = 000001db 537dece8 19b7f70f 555a67c4 27a8cd9b f18aeb9b 56e0c110 56fae6a3

# SCAPI Installation

1. Copy the dlls (JavaInterface.dll, MiraclJavaInterface.dll) to the directory where you run your application.
2. Copy the library “propertiesFiles” with all its content to the running directory.
3. Copy the SCAPI\_Dlog.jar to an installation directory, for example: C:\Program Files\Scapi\Scapi\_Dlog.jar
4. Copy the bcprov-jdk16-141.jar to an installation directory, for example: C:\Program Files\BC\ bcprov-jdk16-141.jar
5. Configure the build path of your project in Eclipse:

In the Libraries tab add two external jars from their installation directory:

Scapi\_Dlog.jar and bcprov-jdk16-141.jar