```
/**
*
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
*
* Copyright (c) 2012 - SCAPI (http://crypto.biu.ac.il/scapi)
* DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
*
* Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"),
* to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense,
* and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
* FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
* WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*
* We request that any publication and/or code referring to and/or based on SCAPI contain an appropriate citation to SCAPI, including a reference to
* http://crypto.biu.ac.il/SCAPI.
*
* SCAPI uses Crypto++, Miracl, NTL and Bouncy Castle. Please see these projects for any further licensing issues.
*
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
*
*/

package edu.biu.scapi.tests.midLayer.cramerShoup;



import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.security.InvalidKeyException;
import java.security.KeyException;
import java.security.KeyPair;
```

```
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

import edu.biu.scapi.exceptions.FactoriesException;
import
edu.biu.scapi.midLayer.asymmetricCrypto.encryption.ScCramerShoupDDHOnGroupElement;
import edu.biu.scapi.midLayer.ciphertext.AsymmetricCiphertext;
import edu.biu.scapi.midLayer.plaintext.GroupElementPlaintext;
import edu.biu.scapi.midLayer.plaintext.Plaintext;
import edu.biu.scapi.primitives.dlog.DlogGroup;
import edu.biu.scapi.primitives.dlog.GroupElement;
import edu.biu.scapi.primitives.hash.CryptographicHash;
import edu.biu.scapi.tools.Factories.CryptographicHashFactory;
import edu.biu.scapi.tools.Factories.DlogGroupFactory;
```

/*This is an application that does the following:

a. For different curves (which are read from a configuration file) and with Zp (with a 1024 bit prime and a 2048 bit prime), and with SHA1 for the elliptic curve instantiations and SHA256 with the Zp instantiations:

- **i. Initializes Cramer Shoup**
- **ii. Generates a group element**
- **iii. Encrypts N times; outputs average encryption time (saying which instantiation)**
- **iv. Decrypts N times; outputs average decryption time (saying which instantiation)**

***/**

```
public class CramerShoupTest {

	private static final String FILES_PATH = System.getProperty("user.dir") +
"/JavaSrc/edu/biu/scapi/tests/midLayer/cramerShoup/";

	static public String runTest(CramerShoupTestConfig config) throws FactoriesException{

	DlogGroup dlogGroup;

	//Create the requested Dlog Group object. Do this via the factory.
	if(config.getDlogProvider() != null){
		dlogGroup=DlogGroupFactory.getInstance().getObject(config.dlogGroup+"("+co
		nfig.algorithmParameterSpec+")", config.getDlogProvider());
	}else {
		config.setDlogProvider("Default");
		dlogGroup=DlogGroupFactory.getInstance().getObject(config.dlogGroup+"("+co
		nfig.algorithmParameterSpec+")");
	}
```

```
//Create the requested hash. Do this via the factory.
CryptographicHash hash;
if(config.getHashProvider() != null){
        hash = CryptographicHashFactory.getInstance().getObject(config.hash,
        config.getHashProvider());
}else {
        config.setHashProvider("Default");
        hash = CryptographicHashFactory.getInstance().getObject(config.hash);
}

//Create a random group element. This element will be encrypted several times as
specified in configuration file and decrypted several times as specified in configuration file.
GroupElement gEl = dlogGroup.createRandomElement();

//Create a Cramer Shoup Encryption-Decryption object. This is done directly by calling
the relevant constructor.
ScCramerShoupDDHOnGroupElement enc = new
ScCramerShoupDDHOnGroupElement(dlogGroup, hash);

//Generate and set a suitable key.
KeyPair keyPair = enc.generateKey();
try {
        enc.setKey(keyPair.getPublic(),keyPair.getPrivate());
} catch (InvalidKeyException e) {
        e.printStackTrace();
}


//Wrap the group element we want to encrypt with a Plaintext object.
Plaintext plainText = new GroupElementPlaintext(gEl);
AsymmetricCiphertext cipher = null;

//Measure the time it takes to encrypt each time.
long allTimes = 0;
long start = System.currentTimeMillis();
long stop = 0;
long duration = 0;

int encTestTimes = new Integer(config.numTimesToEnc).intValue();
for(int i = 0; i < encTestTimes; i++){
        //The actual encryption takes place here
        cipher = enc.encrypt(plainText);
        stop = System.currentTimeMillis();
        duration = stop - start;
        start = stop;
        allTimes += duration;
}
```

```
		//Calculate and output the average running time.
		double encAvgTime = (double)allTimes/(double)encTestTimes;

		GroupElementPlaintext decrypted = null;
		allTimes = 0;
		int decTestTimes = new Integer(config.numTimesToDec).intValue();

		//Measure the time it takes to decrypt each time.
		for(int i = 0; i < decTestTimes; i++){
			try {
				//The actual decryption takes place here
				decrypted = (GroupElementPlaintext) enc.decrypt(cipher);
				stop = System.currentTimeMillis();
				duration = stop - start;
				start = stop;
				allTimes += duration;
			} catch (KeyException e) {
				e.printStackTrace();
			}
		}

		//Sanity check that that the decrypted element equals the original element.
		boolean equal = gEl.equals(decrypted.getElement());

		// Calculate and output the average running time.
		double decAvgTime = (double)allTimes/(double)decTestTimes;

		//Prepare an output string (csv format)
		String result = config.dlogGroup + "," + config.getDlogProvider() + "," +
config.algorithmParameterSpec + "," + config.hash + "," + config.getHashProvider() + "," +
config.numTimesToEnc;

		result += "," + encAvgTime + "," + config.numTimesToDec + "," + decAvgTime + "," +
equal;

		return result;
}
```

```
//Function that reads the configuration file into an array of CramerShoupTestConfig instances.
static CramerShoupTestConfig[] readConfigFile() {
	CramerShoupTestConfig[] configArray = null;
	try {
		BufferedReader bf = new BufferedReader(new FileReader(FILES_PATH +
"CramerShoupTestConfig.ini"));
		String line;
		String[] tokens;
		line = bf.readLine();
		int numOfTests = 0;
		if (line.startsWith("NumOfTests")) {
			tokens = line.split("=");
			String tok = tokens[1].trim();
			numOfTests = new Integer(tok).intValue();
		}
		configArray = new CramerShoupTestConfig[numOfTests];
		int i = 0;
		String dlogGroup = null;
		String dlogProvider = null;
		String algorithmParameterSpec = null;
		String hash = null;
		String hashProvider = null;
		String numTimesToEnc = null;
		String numTimesToDec = null;

		int count = 0;
		while ((line = bf.readLine()) != null) {
			if (line.startsWith("dlogGroup")) {
				tokens = line.split("=");
				dlogGroup = tokens[1].trim();
			} else if (line.startsWith("dlogProvider")) {
				tokens = line.split("=");
				if(tokens.length > 1){
					dlogProvider = tokens[1].trim();
				}
			} else if (line.startsWith("algorithmParameterSpec")) {
				tokens = line.split("=");
				algorithmParameterSpec = tokens[1].trim();
			} else if (line.startsWith("hash")) {
				tokens = line.split("=");
				hash = tokens[1].trim();
			} else if (line.startsWith("providerHash")) {
				tokens = line.split("=");
				if(tokens.length > 1){
					hashProvider = tokens[1].trim();
				}
			} else if (line.startsWith("numTimesToEnc")) {
```

```
                    tokens = line.split("=");
                    numTimesToEnc = tokens[1].trim();
                } else if (line.startsWith("numTimesToDec")) {
                    tokens = line.split("=");
                    numTimesToDec = tokens[1].trim();
                }


                count++;
                if (count == 7) {
                    configArray[i] = new CramerShoupTestConfig(dlogGroup,
                    dlogProvider, algorithmParameterSpec, hash, hashProvider,
                    numTimesToEnc, numTimesToDec);

                    i++;
                    count = 0;
                }
            }
            //Finished reading, close the file
                bf.close();
            } catch (IOException e) {
                System.err.println(e.getMessage());
            }
            return configArray;

}
```

```
/**
* This program tests the average running times of encrypting and decrypting a Group Element
* with the Cramer Shoup encryption scheme.
* It reads a set a tests from a config files, runs them and prints the results. The set of tests
* contains information about the Dlog Group to use,
* the Hash function to use (it is possible to choose the providers for them).
 * @param args
 * @throws FactoriesException
 */

public static void main(String[] args) throws FactoriesException {
	try {
		// Get parameters from config file:
		CramerShoupTestConfig[] config = readConfigFile();
		DateFormat dateFormat = new
SimpleDateFormat("dd_MM_yyyy_HH_mm_ss");
		Date date = new Date();
		String testName = FILES_PATH + "CramerShoupTestResults_" +
dateFormat.format(date) + ".csv";
		PrintWriter out = new PrintWriter(testName);
		out.println("Dlog Group,Dlog Provider,Dlog Parameter,Hash,Hash
Provider,Number of Times Encrypting, Average Encrypting Time (ms),Number of Times
Decrypting,Average Decrypting Time (ms),Decrypted Element Equals Plaintext");
		out.flush();
		String result = null;
		//Run all the tests stipulated in the configuration file
		for (int i = 0; i < config.length; i++) {
			result = runTest(config[i]);
			out.println(result);
			System.out.println(result);
		}
		//Close the file
		out.flush();
		out.close();
	} catch (IllegalArgumentException e) {
		e.printStackTrace();
	} catch (IOException e) {
			e.printStackTrace();
	} catch (FactoriesException e) {
		e.printStackTrace();
	}
}

}
```