CSC318: Cryptography and IT-Security – Coursework 2

Phillip James

Due: 11:00am, 15th March 2022

This coursework involves programming and Merkle's puzzles. There is a total of 25 marks available for this coursework and it is worth 10% of the module. On Canvas as part of the assignment you will find a zip file containing Java code (some of which has been obfuscated) to be used when completing the coursework. Please also note that this coursework will be partially accessed through automated testing (details below). Please do not place your code in a package (it must use the default package). Please note that I have created all the code and tests using Java 8. I recommend you use this version of Java for you submission.

The coursework is based around the following scheme:

> *Dear Bob,*
>
> *I am going to send you* 4096 *puzzles. Each puzzle is a cryptogram whose plaintext starts out with* 128 *zero bits (*16*-bytes), followed by a* 16*-bit (*2*-byte) puzzle number in the range 1 to 4096, and then a* 64*-bit (*8*-byte) key. Each cryptogram has been encrypted using DES with a different key specification whose final* 48 *bits are zeros.*
>
> *Please pick one cryptogram at random and break it by brute force, trying all* 2^{16} *keys ending in* 48 *zeros. You know you have broken it when you find a key that yields a plaintext starting with* 128 *zero bits. After breaking the cryptogram, extract the* 64*-bit key and use it as our shared key. As your first message send me back the puzzle number in plaintext, so I know which key you are going to use.*
>
> *Yours sincerely, Alice*

Part 1: Implement Puzzles [4 Marks]

Write a *Puzzle* class for storing a puzzle. This must have at least the following public methods:

- A constructor that takes a puzzle number as an int and a secret key as a SecretKey object and then constructs a Puzzle object.
- A method *getPuzzleNumber* that returns the puzzles number as an int.
- A method *getKey* that returns the puzzles key as a SecretKey.
- A method *getPuzzleAsBytes* that returns a byte array representing the puzzle in the above specified format.

You can use tests one through to four to check your code for this part of the coursework.

Part 2: Implement Puzzle Generation and Encryption [15 Marks]

Next write a *PuzzleCreator* class that contains at least the following public methods:

- A constructor that takes no parameters.

- A *createPuzzles* method that generates and returns an ArrayList⟨Puzzle⟩ of 4096 Puzzle objects.
- A *createRandomKey* method that returns a byte array that can be used to form a DES key. This byte array should be in the above specified format (final 48 bits should be zeros).
- An *encryptPuzzle* method that takes a byte array representing a key and a puzzle object and encrypts the puzzles byte representation into a byte array representing the encrypted puzzle. The method should return this byte array. Note that if completed correctly the resulting byte array will be 32 bytes in length.
- An *encryptPuzzlesToFile* method that takes as a parameter a String representing a filename (e.g. "puzzles.bin") and encrypts all 4096 puzzles and also writes them to a binary file (the byte data as it is, no spaces or new lines) with given name.
- A *findKey* method that take as a parameter an int representing a puzzle number and returns the key for that puzzle as a SecretKey object.

You can use tests five through to nineteen to check your code for this part of the coursework.

Part 3: Demonstrate Merkle's Puzzles Working [3 Marks]

Finally, you should demonstrate that you can use your code to complete the Merkles puzzle communication. You have been given a PuzzleCracker class that can perform the required functions of Bob. In particular, the PuzzleCracker class contains the methods:

- A constructor for *PuzzleCracker* that takes as a parameter the filename to crack as a String.
- A method *crack* that takes as a parameter an integer representing the puzzle to crack (note this is not the puzzle number, but a number representing a puzzle in the order that Bob has them in given encrypted file). This method will return a Puzzle object.
- A method *decryptMessage* that given a byte array will print a String representing a decryption of that string using the key from the puzzle that has been cracked.

To use this code, you will need to include the class file when compiling. Appendix B gives some guidance on how to include class files into the compilation process for various IDE's.

You should create a class called *Merkle* that contains a main function that:

1. Use your *PuzzleCreator* code to create an encrypted puzzle file representing Alice's puzzles.
2. Use the *PuzzleCracker* class to crack a random puzzle (I will call this Bob's puzzle from now on).
3. Once cracked, use the *findKey* method you have implemented to find the puzzle in Alice's collection of puzzles that had the same puzzle number as Bobs puzzle.
4. Use the key from Alice's puzzle to encrypt the message "Testing Merkles Puzzles!", before calling the *decryptMessage* function from the *PuzzleCracker* class with the encrypted message as a parameter. This call will cause the message to be decrypted with Bobs cracked puzzle key and print the result to the screen.

Feel free to print whatever output you want for representing these steps.

Part 4: Programming Style and Documentation [3 Marks]

You should use suitable variable names, methods and comments. You should think throughout about security of the scheme. Finally, you should include suitable JavaDoc for all your methods and classes.

Hints and Tips

You have been given a file named "CryptoLib.java". This file contains a number of helper methods that you may want to use. In particular it contains:

- methods for converting small integers (up to $65,535$) to byte arrays and back.
- a method for converting byte arrays into SecretKeys.
- a method for converting byte arrays to Hex strings.

Along with this, you may (or may not) find the Java classes FileInputStream and FileOutputStream useful for reading and writing binary files.

Automated Testing

As this coursework will be partially automatically marked. To give re-assurance that you have completed sections of the coursework correctly, I have given you access to a subset of the tests that will be used for automated testing in Appendix A. You can check how your code performs against these tests by making a submission to Autograder (see submission section below).

PLEASE NOTE: the online tests will likely fail if:

- **Any of your methods throw (checked) exceptions, as your tests are pre-compiled and hence these check will not happen during compilation.**
- **Any of your code (apart from the main method) print output. As the online tests will check against output generated by the test (which, for each test, should only be the output given in Appendix A).**

Submission

You must submit your code files (not zipped) to the testing platform available at:

`csautograder.swansea.ac.uk`

You can log in to the system using you normal university credentials. If you experience a 502 error when trying to login, you will need to delete your cookies (or use an incognito window) and re-try, things should then work.

This system will run all of the test cases from Appendix A (and more tests for marking purposes) and display to you the output of these tests. Please make sure these outputs match your expectations. Multiple submissions will be accepted, although only **your last submission will be marked**.

You should submit your work before **11:00am, 15th March 2022**.

PLEASE NOTE:

- **I recommend you try out submitting some code to csautograder as soon as possible to familiarise yourself with how the system works.**
- **csautograder will place your submission in a queue to be tested. Thus it is recommended you do not leave it until the last moment to check your code as if the queue grows large, the time for testing could grow to over 1 hour.**
- **All submissions uploaded before 11am (even if not run) will be accepted.**

By submitting coursework, electronically and/or hardcopy, you state that you fully understand and are complying with the university's policy on Academic Integrity and Academic Misconduct. The policy can be found at:

`https://myuni.swansea.ac.uk/academic-life/academic-misconduct`

In the event that you think you think you witness plagiarism you should get in touch with your Academic Mentor straight away and explain the issue to them.

Appendix A

Test	Description	Expected Output
1	This test will call your Puzzle constructor followed by the method getPuzzleAsBytes. It will then check the length of the returned byte array	26
2	This test will call your Puzzle constructor followed by the method getPuzzleAsBytes. It will then check that the returned byte array is of correct format (16 bytes zeros followed by 10 bytes that are not zeros)	true
3	This test will call your Puzzle constructor followed by the method getPuzzleAsBytes. It will then check that the puzzle byte data correctly models the given puzzle number and key	true
4	(Not given, used for marking only)	–
5	This test will call your PuzzleCreator constructor followed by the method createPuzzles and will check that an ArrayList of 4096 items is returned.	4096
6	This test will call your PuzzleCreator constructor followed by the method createPuzzles and will check that each of the elements within the returned array list is a valid puzzle instance	true
7	(Not given, used for marking only)	–
8	(Not given, used for marking only)	–
9	This test will call your PuzzleCreator constructor followed by the method createRandomKey and will check that the size of the returned byte array is 8	8
10	This test will call your PuzzleCreator constructor followed by the method createRandomKey and will check that the returned byte array ends in 6 bytes of zeros	true
11	(Not given, used for marking only)	–
12	This test will call your PuzzleCreator constructor followed by the method createRandomKey. It will then use the returned random key as a puzzle object as parameters to call encryptPuzzle. Finally it will check that the returned byte array representing the encrypted puzzle is 32 bytes long	32
13	This test will call your PuzzleCreator constructor followed by the method createRandomKey. It will then use the returned random key as a puzzle object as parameters to call encryptPuzzle. Finally it will check that the byte array representing the encrypted puzzle, when decrypted and interpreted as a puzzle, gives the correct puzzle number	true
14	(Not given, used for marking only)	–
15	This test will call your PuzzleCreator constructor followed by the method createPuzzles. It will then call the method encryptPuzzlesToFile with the filename “puzzles.bin”. Next it will read in first 32 bytes of data in the written file, crack the decryption key and check the decrypted data can be used to form a puzzle whose puzzle number is in the range 1-4096	true
16	This test will call your PuzzleCreator constructor followed by the method createPuzzles. It will then call the method encryptPuzzlesToFile with the filename “puzzles.bin”. Next it will read in first 32 bytes of data in the written file, crack the decryption key and check the decrypted data can be used to form a puzzle where the key obtained is the correct key according to the puzzle creator	true
17	(Not given, used for marking only)	–
18	This will call your PuzzleCreator constructor followed by the method createPuzzles. It will then look up the puzzle number for the first puzzle. Next it will call your your findKey with this puzzle number and will and check that the key of the returned puzzle matches the key of the first puzzle	true
19	(Not given, used for marking only)	–
20	(Not given, used for marking only)	–

Appendix B

Below are instructions on including the given class files within your build path.

No IDE:
Just include the class files in the same folder as your code files and compilation should work.

Eclipse:
Right click on your top level project folder and select:

Build Path → Configure Build Path → Add External Class Folder.

Then select the folder which contains the class files you would like to include and click open.

Intellij:
Open the project structure dialog with:

File → Project Structure

then select:

Project Settings → Modules

on the left. Then select the module whose classpath you want to modify. On the Dependencies tab on the right side of the dialog you can add the directory containing your class files.