

# Task 1: Session Keys

**Due** 23 Nov 2022 by 17:00      **Points** 0

**Submitting** a text entry box or a website url

**Available** 10 Nov 2022 at 17:00 - 23 Nov 2022 at 17:00

This assignment was locked 23 Nov 2022 at 17:00.


## The SessionKey class

This task is about creating encryption keys, and converting the keys to a portable format that can, for instance, be communicated over the network. You will create a class `SessionKey`, which will be used in the final project assignment as a session key to encrypt traffic in a communication session.

There are two constructors for `SessionKey`:

```
SessionKey(Integer keylength)
SessionKey(byte[] keybytes)
```

The first constructor creates a random `SessionKey` of the specified length (in bits). The second variant creates a `SessionKey` from a byte array. The byte array contains an existing key, represented as a sequence of bytes (more about this later).

The actual encryption key that is created is a symmetric key (or secret key) for AES. JSA and JCE use the class [SecretKey](https://docs.oracle.com/javase/8/docs/api/javax/crypto/SecretKey.html)  (<https://docs.oracle.com/javase/8/docs/api/javax/crypto/SecretKey.html>) to represent symmetric keys. Therefore, it is a good idea to use the `SecretKey` class internally in your `SessionKey` as well.

In order to retrieve the `SecretKey` from a `SessionKey` object, there is a `getSecretKey` method to retrieve the corresponding `SecretKey`:

```
SecretKey getSecretKey()
```

In the final part of the project assignment, you will create session keys and transmit them over the network. But we cannot just take a Java object and transmit it over the network – what happens if the other party is not implemented in Java? Moreover, we do not want to transmit a symmetric key in cleartext over the network – that would put the confidentiality of the key at serious risk.

This means that we should encrypt the symmetric key before we transmit it over the network. Encryption is an operation that works on binary data – it takes a sequence of bytes and transforms it to another sequence of bytes. Hence, in order to encrypt our symmetric key, we need to first convert it to a sequence of bytes.

So, long story short, the `SessionKey` class should export a key as a sequence of bytes. For this, we use the `getKeyBytes` method:

```
byte[] getKeyBytes()
```

## Assignment

Implement the `SessionKey` class in Java, with the methods and constructors described above.

To help you get started, there is a file with a skeleton declaration of the `SessionKey` class in the project repository on KTH GitHub. Use that file as the basis for your solution, and fill in the code.

## Key Randomness


The quality of encryption depends a lot on the quality of the keys. What does that mean? How can you check that the key that you have generated is a "good" key? Presumably a key is generated as a random key. What constitutes a good random number generator is something that researchers have given a good deal of thought.

Surely you cannot look at a single number and decide if it is random or not? But think of a key not as a single number, but as a sequence of bits, where you decide the value of each bit by flipping a coin. So a 256-bit key is the result of

flipping a coin 256 times. Then you might be able to do some analysis to determine the quality of the coin-flipping process.

On a lighter note, you may get some inspiration from these comic strips:

- <http://www.dilbert.com/strip/2001-10-25>   
(<http://www.dilbert.com/strip/2001-10-25>)
- <https://xkcd.com/221>  (<https://xkcd.com/221/>)

You may also find this website interesting; it is about using atmospheric noise to achieve randomness: <https://www.random.org/randomness>.   
(<https://www.random.org/randomness/>)

## Resources

There is a personal repository created for you on KTH GitHub with skeleton code and test programs. The repository is called "<username>-SessionKey", where "<username>" is your KTH user name.

Clone the repository and use it as a starting point for your work.

## Testing

How can you test that your code works correctly? If it doesn't work as it should, you will get into debugging problems in the final assignment, so it is much better to spend some time now making sure that your code does what it is supposed to do.

Here are some suggestions for testing:

You could examine the key that is created. It should consist of 128 bits (or 192 or 256, depending on what key length you used). Print out the key (which is a byte array). Check that the output is indeed 128 (192, 256) bits, and that the bits appear random.

## Unit Tests

We have put together a small set of basic tests that you can run on your code. It will help you check the basic functionality, and also to verify that the code can be

integrated in a larger environment. There is a test class, `SessionKeyTest`, which uses your definition of the `SessionKey` class, and contains a number of test.

The tests are done for the JUnit test framework for Java. You can find the tests in your personal repository on KTH GitHub.

## Submission

In order to complete your submission, you need to do two things:

- Commit your changes in git and push to your personal repository on KTH GitHub.
  - **Note:** the file should not contain any package declarations. If your IDE inserts package declarations for you, you need to remove them.
- Submit the URL of your personal KTH GitHub repository here, in this assignment, to notify us about the submission.
  - The URL looks something like this: "https://gits-15.sys.kth.se/IK2206HT22/<user>-SessionKey.git"

**Important:** We will not automatically check your personal KTH GitHub repository for updates. Whenever you make an update that you want us know about, you have to make a submission here. Submit the URL of your personal KTH GitHub repository, as described above.