NOTE: SCENARIO IS WHAT THE PUZZLE TAKER SEES.

SCENARIO:

You are developing a secure texting system. Text messages must be encrypted before being transferred over the network. The `encrypt` method takes two arguments: `text`, a `String` value to be encrypted, and `cipher`, a properly initialized `Cipher` object to process the encryption. The method returns a `byte` array representing the ciphertext of the original (plain) text. Consider the snippet of code below and answer the following questions, assuming that the code has all required permissions to execute.

Code:

```
01   // OMITTED: Import whatever is needed.
02   public final class CryptoUtils {
03     public static byte[] encrypt (String text, Cipher cipher)
04         throws IOException {
05       byte[] bytes = text.getBytes();
06       try (
07         InputStream input = new ByteArrayInputStream(bytes);
08         ByteArrayOutputStream output = new ByteArrayOutputStream();
09         OutputStream proc = new CipherOutputStream(output, cipher)
10       ) {
11         // The following statement reads all bytes from
12         // 'input' stream and writes them to 'proc' stream.
13         IOUtils.copy(input, proc);
14
15         // Return the ciphertext (as a byte array).
16         return output.toByteArray();
17       }
18     }
19   }
```

Questions:

1. What will the `encrypt` method do when executed?

2. Which one statement is correct if the `encrypt` method gets called to encrypt an arbitrary (non-null, non-empty) text with an arbitrary (properly initialized) `Cipher` object?

a. The method always encrypts the text correctly.
b. The method never encrypts the text correctly, and it throws an exception.
c. The method never encrypts the text correctly, but it does not throw an exception.
d. Depending on the `cipher` object, the method may encrypt the text correctly.
e. None of the above.

[Other statistical questions will be imported here while creating the survey.]

NOTE: ANSWER IS TO BE SHOWN TO THE PUZZLE TAKER AT THE END OF THE SESSION.

ANSWER:
d
According to the API documentation, while using `CipherOutputStream` or `CipherInputStream` object to finalize the (encryption or decryption) process, the `close` method should be called on the stream object. If not, the produced data (plaintext or ciphertext) is invalid. This implementation is using the try-with-resource feature of Java (lines 6~9). As a result, all resources (in this case streams) defined in the initialization statement (lines 6~9) are going to be closed right after the execution of the last statement of the `try` block (line 16). That means the `CipherOutputStream` object is going to be closed after collecting the produced data (line 16 by calling the `toByteArray` method). Thus, even though the method does not throw any exception, the produced data (in this case ciphertext) is invalid and cannot be decrypted to the original text. Depending on how the `cipher` object is initialized, this code might work in some cases, though it is not generally the correct way of using cipher streams. To fix the code the `output` variable should be taken out of the `try` block, and the `toByteArray` method should be called right after the `try` block ends.

NOTE: THE REST OF THE DOCUMENT CONTAINS EXTRA INFORMATION FOR THE PROJECT RESEARCHERS. IT IS NOT TO BE SHOWN TO THE PUZZLE TAKERS.

TAGS:
java, cryptography, cipher-streams, finalization, api-protocol-usage

CATEGORIES:
Blindspot - YES
Type - Crypto
Number of distinct functions - 5
Number of total functions - 5
Blindspot function - `InputStream.close()`
Function call omitted - YES
Blindspot type - Omitted function call
Number of Parameters in the blindspot function - 0 parameters
Cyclomatic complexity - 2

NAME:
CipherInputStream, CipherOutputStream - Filter streams to process (read or write) the data before doing the main IO operations (read and write)

DESCRIPTION:

A `CipherOutputStream` is composed of an `OutputStream` and a `Cipher` so that the `write` method processes the data before writing it out to the underlying `OutputStream`. The `cipher` argument must be fully initialized before being used by a `CipherOutputStream`.
A `CipherInputStream` is composed of an `InputStream` and a `Cipher` so that `read` method return data that is read in from the underlying `InputStream` has been further processed by the `Cipher`. The `cipher` argument must be fully initialized before being used by a `CipherInputStream`.
After finishing the process (reading or writing data from or into the stream), but before fetching the final result, the stream has to be closed.

BLINDSPOT:
The API usage protocol specified by the Java standard library (i.e. how to use cipher streams) may cause a blindspot, especially because, in some cases, violating the protocol will have no implication. In addition, codes violating the protocol do not cause any error or exception. Instead, they silently produce invalid data. For example, if one tries to encrypt a plaintext using the given code in the scenario, the code generates an invalid ciphertext, meaning the generated ciphertext cannot be decrypted to the original plaintext.

CORRECT USE EXAMPLE:

Code:
```
import java.io.*;
import java.security.*;
import javax.crypto.*;
public class CryptoUtils {
  public static byte[] encrypt ( Cipher cipher, byte[] plain )
      throws IOException {
    ByteArrayInputStream input = new ByteArrayInputStream(plain);
    ByteArrayOutputStream output = new ByteArrayOutputStream();
    OutputStream processor = new CipherOutputStream(output, cipher);
    IOUtils.copy(input, processor);

    processor.close();

    return output.toByteArray();
  }

  public static byte[] decrypt ( Cipher cipher, byte[] encrypted )
      throws IOException {
    ByteArrayInputStream input = new ByteArrayInputStream(encrypted);
    ByteArrayOutputStream output = new ByteArrayOutputStream();
    InputStream processor = new CipherInputStream(input, cipher);
    IOUtils.copy(processor, output);

    processor.close();
```

```
    return output.toByteArray();
  }
}
```

MORE INFORMATION:
To see the correct way to use the API, look at the JX24-cipher-streams-close puzzle.

REFERENCES:
1. CipherInputStream#close
2. CipherOutputStream#close