CIS551: Networks and Security	Project 3
Spring 2009	Due: 6 April 2009 at 11:59 PM

Cryptanalysis

1 SDES block cipher

In this project, you will implement a simplified version of the DES block cipher algorithm. Naturally enough, it is called SDES, and it is designed to have the features of the DES algorithm but scaled down so it is more tractable to understand. (Note however, that SDES is in no way secure and should not be used for serious cryptographic applications.)

The photocopied handouts and the web page linked to from the course home pages that accompany this project description give the detailed specifications of SDES, including a sample Java implementation that uses a *different* representation (see below). For the purposes of this project, you are *strongly encouraged* to complete as much of your implementation as you can *without* refering to the given Java implementation—the programming portion of this project isn't that difficult; the point is for you to closely examine the steps of a typical block cipher implementation. (See the note on academic integrity at the end of this document.)

SDES encryption takes a 10 bit raw key (from which two 8 bit keys are generated as described in the handout) and encrypts an 8 bit plaintext to produce an 8 bit ciphertext. Implement the SDES algorithm in a class called SDES. The encryption and decryption methods should match the interface below:

```
public static byte[] Encrypt(byte[] rawkey, byte[] plaintext)
public static byte[] Decrypt(byte[] rawkey, byte[] ciphertext)
```

Here, rather than compactly representing the SDES plaintext and ciphertext using byte-sized (8-bit) variables or integers, the data is represented using byte arrays of length 8. Similarly the 10 bit keys are represented as arrays of length 10. Although this design is extremely inefficient (it uses 8 times more space), it forces you to understand the encryption steps more explicitly, and you can easily experiment with the results. For example, one might declare a 10-bit raw key in a test program like this:

```
byte key1[] = {1, 1, 1, 0, 0, 0, 1, 1, 1, 0};
```

Note that this representation is different from the sample Java implementation available on the web pages, which makes using that code directly for this project difficult.

To verify that your implementation of SDES is correct, try the following test cases:

Raw Key	Plaintext	Ciphertext
0000000000	10101010	00010001
1110001110	10101010	11001010
1110001110	01010101	01110000
1111111111	10101010	00000100

Use your implementation to complete the following table:

Raw Key	Plaintext	Ciphertext
0000000000	00000000	?
1111111111	11111111	?
0000011111	00000000	?
0000011111	11111111	?
1000101110	?	00011100
1000101110	?	11000010
0010011111	?	10011101
0010011111	?	10010000

1.1 TripleSDES

The DES algorithm uses keys of length 56 bits, which, when DES was originally designed, was thought to be secure enough to meet most needs. However, due to Moore's law, the increase in computing power makes it more tractible to brute-force crack a 56-bit key. Thus, an alternative version of DES using longer keys was desirable. The result, known as Triple DES uses two 56-bit raw keys k_1 and k_2 and is implemented by composing DES with itself three times in the following way:

$$E_{3DES}(p) = E_{DES}(k_1, D_{DES}(k_2, E_{DES}(k_1, p)))$$

Here, p is the plaintext to encrypt, E_{DES} is the usual DES encryption algorithm and D_{DES} is the DES decryption algorithm. This strategy doubles the number of bits in the key, at the expense of perfoming three times as many calculations. (You might wonder why the algorithm is not just $E_{DES}(k_1, E_{DES}(k_2, p))$. This approach was shown to offer only the security of a 57-bit key rather than 112 bits as you might expect.)

The TripleDES decryption algorithm is just the reverse:

$$D_{3DES}(c) = D_{DES}(k_1, E_{DES}(k_2, D_{DES}(k_1, c)))$$

For this part of the project, implement a class called TripleSDES that provides the following methods and calculates the TripleSDES encryption.

```
public static byte[] Encrypt( byte[] rawkey1, byte[] rawkey2, byte[] plaintext )
public static byte[] Decrypt( byte[] rawkey1, byte[] rawkey2, byte[] ciphertext )
```

Use your implementation to complete the following table:

Raw Key 1	Raw Key 2	Plaintext	Ciphertext
0000000000	0000000000	00000000	?
1000101110	0110101110	11010111	?
1000101110	0110101110	10101010	?
1111111111	1111111111	10101010	?
1000101110	0110101110	?	11100110
1011101111	0110101110	?	01010000
0000000000	0000000000	?	10000000
1111111111	1111111111	?	10010010

1.2 Cracking SDES and TripleSDES

For this part of the project, you will use your SDES imlementation and brute-force search to crack some encoded English messages. This would be quite straightforward if the input text used standard ASCII encodings, because you can test each key to see if it generates output that is purely alphanumeric (almost all of the wrong outputs will contain random ASCII gibberish). To make the problem more interesting, the text in the messages here are encoded using Compact ASCII, or CASCII for short. Also, a little bit of "noise" was introduced into the source text.

CASCII characters are 5 bits long, which gives just enough space for the upper case letters and some punctuation: 0 = ' ' (space), 1–26 = 'A'–'Z', 27=',', 28 = '?', 29=':', 30='.', 31 = '''. The file `CASCII.java` available on the course web site provides a definition of CASCII constants and some useful conversion functions. See the file comments for details. Although it should not affect the code you write for the project, CASCII uses big-endian encodings. For example, the letter 'T' = 20 is represented by the bit sequence 00101. This may be useful when debugging your programs.

Since SDES and TripleSDES only work on blocks of size 8 bits, when converting a CASCII string it is necessary to pad the bit representation with 0's to obtain a multiple of 8. See the `convert` and `toString` methods in `CASCII.java`.

1. Give the SDES encoding of the following CASCII plaintext using the key 0111001101. (The answer is 64 bits long.)

   ```
   CRYPTOGRAPHY
   ```

2. The message in the file `msg1.txt` was encoded using SDES. Decrypt it, and find the 10-bit raw key used for its encryption.

3. The mesage in the file `msg2.txt` was encoded using TripleSDES. Decrypt it, and find the two 10-bit raw keys used for its encryption.

1.3 Hints

There are only 1024 10-bit keys, so simply trying them all and looking at the output might be a reasonable way to crack SDES. However, there are more than a million choices of two 10-bit keys, so looking through the output of each TripleSDES key choice by hand is infeasible. Also, keep in mind that it will take roughly 3000 times longer to search the The problem is knowing when you've found the correct decoding of the ciphertext—part of the difficulty of cryptanalysis is knowing when you have succeeded!

2 What to turn in

Each group should use `turnin` to submit the following items (one submission per group).

1. A project report containing the names of the students in your group. In addition to the items specified below, include a one- or two-sentence description of each group member's contributions to the project and an estimate of the number of hours each member contributed.

   Your group will receive a single grade for the project. The information about individual contributions will be used at the end of the semester to determine borderline grade cases.

2. Solutions for the test cases described in Sections 1.1 and 1.2, the bits making up the keys of the SDES and TripleSDES encrypted messages, and the messages themselves. Also describe the filtering strategy you used to know that the keys are correct.

3. Your source code. This should be single directory containing a file called `README` that describes the contents of the directory and any special instructions needed to run your programs (i.e. describe any command-line arguments). Your SDES implementation should be completed using well documented Java, and the classes `SDES` and `TripleSDES` should be implemented according to the interface described above. We will test your code on Eniac.

4. Note: if your group *did* resort to using the sample imlementation during your project development, please indicate this fact clearly in your documentation.

3 A Word about Academic Integrity

The Simplified DES project has been used at many universities and in prior courses here at Penn, because it is a well designed example of shared key cryptography. As a consequence, there are many implementations of SDES available on the internet, including the sample implementation referenced by the course web site. For the purposes of this project, it is OK to look at that implementation as a last resort (though it really shouldn't be necessary), but you *must* create your complete implementation yourself. *Do not copy code from the example or any other implementation.* Our course staff is familiar with many solutions available on the web and we have a repository of all the old solutions submited by Penn Students. We *will* look for cases of cheating and treat them as academic misconduct. Similarly, collaboration between groups is prohibited and will be treated as a serious violation of the University's academic integrity code.