CIS553/753 Applied Cryptography, Spring 2014
Due: March 25

Implement Double DES, and meet-in-the-middle attack

Your job of this project is to implement Double DES based on a single-DES implementation, and then carry out a meet-in-the-middle attack to recover a partially known key, given a plaintext-ciphertext pair.

Part 1: Implement Double DES (10 points)

Implement Double DES, whose key size is 112 bits, based on the single-DES implementation provided by a crypto API of the programming language of your choice. You should read the manual of the API to learn how to encrypt/decrypt in DES. One objective of this project is to get you familiarized with using library API's to conduct crypto operations.

Your program should take a key and a plaintext as arguments, and print on the screen the ciphertext. All the strings in input and output are interpreted as hexadecimal representations. Each hexadecimal digit represents four binary digits. So the key should have 28 hex digits (equivalent to 112 bits). Please use upper-case letters for the hex digits in the key and ciphertext, and lower-case letters for the hex digits in the plaintext.

For example, suppose your program name is "double-des". Given

```
double-des 00000000000000011111111111111 0123456789abcdef
```

the output should be

```
3057B90BD52BAE5E
```

As another example, given

```
double-des 0123456789ABCDEF0123456789AB 6162636461626364
```

the output should be

```
8FFAB5BBCF35B580
```

When using a block cipher like DES, we also need to specify cipher modes and padding modes. Both phases of your Double DES should use the ECB mode. Both phases of Double DES should use the padding mode "NoPadding". This means the plaintext must have a length of multiple blocks (8 bytes, or 16 hex digits).

DES uses a 56-bit key. However, in standard format the 56-bit key used by DES is stored in eight bytes, in which the least significant bit of each byte is used for odd parity (so that every byte in the key has an odd number of "1" bits). This is why your API function may require a byte array of size 8 for the key. Therefore, your

program may need to add the appropriate parity bit for every 7 bits, to convert a DES key to the format acceptable by the API. Again, read the manual of the API carefully.

Part 2: Carry out a meet-in-the-middle attack (10 points)

You will receive an email from the instructor in which you are given a partially-known key and a plaintext-ciphertext pair. For example, the plaintext could be "48656c6c6f20576f", and the ciphertext is "A95645B5D781BBAF". We know that the ciphertext is encrypted by Double DES (with key size 112 bits). We also have a partial key "????113311441??????AABB2233", where ? stands for an unknown hex digit (4 bits). Figure out the rest of the key using the meet-in-the-middle attack.

Note:

- You better use a hash table to perform efficient middle-value look up.
- If your program is too slow to find the key, you can trade some points for some bits of the key: For each bit we give you, you lose 5 points. Half an hour in Beocat is the maximum time we can accept for your program to find the key.
- The instructor may also test your key-finder on other keys. So make sure your program can take the parital key and plaintext-ciphertext pair from the command-line input. For example:

  ```
  break_2des 48656c6c6f20576f A95645B5D781BBAF ????113311441??????AABB2233
  ```

  the output should be

  ```
  11221133114411556677AABB2233
  ```

What you need to submit

Submit all your work at K-State Online. The submission includes

- A readme file which contains clear instructions on how to compile and run your program.
- All your source code (for both part 1 and part 2). Make sure that your code can be compiled and run on the CIS Linux machines.
- For part 2, a script that is ready to run on Beocat by *any user*. This means you cannot hardcode the program paths in your script since the grader will be logged in as a different user and compile your code at his home directory. Instead, use relative paths in your script. You may lose some points if we find it difficult to make your code run on Beocat.

Some sample tests are provided in the file test2DES; we will run other tests as well.

Beocat Cluster

You will need to use Beocat cluster for part 2. **Please read the documentation page for Beocat cluster carefully before using the system.** Here is an example showing how to use it:

- Log into the Beocat head node. (Type "ssh your-eid@beocat.cis.ksu.edu", and input your eid password when asked.)

- Write the commands you are going to use into one shell script.
- Submit your job through the command line, *e.g.*, "qsub -l mem=4G,h_rt=20:00:00 -pe single 1 YOURJOB.sh".
  This command means you request 1 core for your job, and each of them has 4GB memory. Also the maximum runtime for your job is 20 hours. YOURJOB.sh is the script you need to write in the second step. You can change either of aforementioned parameter to accommodate your job.

You shall always use the qsub command to submit a job to run on Beocat. It is against the Beocat policy to run computation-intensive programs on the head node.

Appendix

Some potentially useful resources:

- **Java SE Security Documentation**
- **OpenSSL crypto library**
- **PyCrypto - The Python Cryptography Toolkit**
- **Invoking CryptoAPI in .NET (C# version)**

IMPORTANT: If you have referred to any existing examples (e.g. online materials), you must cite those examples in your submission. IN NO CASES MAY YOU COPY EXISTING CODE WRITTEN BY OTHERS.

Acknowledgment

We would like to thank Prof. Gang Tan from LeHigh University for providing the original version of this assignment.