University of Maryland
CMSC414 — Computer and Network Security
Professor Jonathan Katz

Problem Set 1

Due by Sept. 22, 11:59 PM

Make sure to **check the HW1 FAQ** (linked from the course webpage) before your final submission: this page will include clarifications and possible hints, format requirements for the various submitted files, and submission instructions.

1. (Warmup.) Create a program to encrypt using the one-time pad. Specifically:
   - Create a program `OTPgen.java` that takes as input a positive integer $n < 500$ (representing the desired message length) and generates a random key of the appropriate length. You should output the key in hexadecimal formal to a file `OTPkey.txt`. (You may assume that n is a multiple of 8.)
   - Create a program `OTPenc.java` that reads a key from `OTPkey.txt` and a message from an ASCII file `OTPmsg.txt` and encrypts the message using the given key. Output the ciphertext in hexadecimal format to a file `OTPctext.txt`. You should return an error if the message and key lengths do not "match".
   - Create a program `OTPdec.java` that reads a key from `OTPkey.txt` and a ciphertext from a hex file `OTPctext.txt` and decrypts the ciphertext using the given key. Output the message in ASCII format to a file `OTPdecrypt.txt`. You should return an error if the ciphertext and key lengths do not "match".

2. You will write a program to perform ECB-, CBC-, and CTR-mode encryption using DES and AES. (Note: you will *write your own program* as explained below, even though the JCE already provides this functionality.)
   - The JCE does not provide a direct way to access the DES or AES block ciphers. (Instead, it only supports calls for doing encryption using these block ciphers.) However, you can access the DES block cipher via:

     ```
     Cipher DEScipher = Cipher.getInstance("DES/ECB/NoPadding");
     ```

     and similarly for AES. **Explain in 1–2 sentences why this gives access to the underlying block cipher**.
   - Create a program called `keygen.java` that takes a single command-line argument ("-DES" or "-AES") indicating the cipher to use. This program should generate a random key and write it to a file called `key.txt` in hexadecimal format. I want you to generate this key yourself using `SecureRandom`, *not* using the built-in method for generating cipher keys.[1]

[1] Recall that a DES key is 64 bits long, yet only 56 of these are random; the remainder are check bits. Make sure that you generate a *random* but valid DES key.

- Create a program `encrypt.java`. This program should take two command-line arguments; the first ("-DES" or "-AES") indicating the cipher, and the second ("-ECB", "-CBC", or "-CTR") indicating the mode. This program should read a key from the hex file `key.txt` and a plaintext from the ASCII file `msg.txt`, and output a ciphertext to the hex file `ctext.txt`. It should output an error if the file length (in bits) is not a multiple of block length of the cipher being used.
- Create a third program `decrypt.java` that takes the same command-line arguments as above, and recovers the plaintext given the key and the ciphertext.

After you have completed the above, please answer the following questions:

(a) How long (in bits) is the ciphertext you generate when using DES in each of the 3 modes you implemented, as a function of the length of the plaintext? What about when using AES?

(b) Compute the fraction of 0-bits in the keys generated by your `keygen.java` program.[2] Generate 1000 keys and compute the average fraction of 0-bits across all these keys. How do the results compare to what you would expect if your key was truly generated at random?

(c) Now run a simple statistical test on *ciphertexts*. Specifically, do the following 1000 times (with a fresh random key each time):

- Encrypt a plaintext file consisting of 6400 A's, and view the resulting ciphertext as a sequence of bytes.
- Find the byte that occurs most often in the ciphertext, and compute the frequency p_{max} with which it occurs.

Tabulate the fraction of times (over your 1000 trials) tha tp_{max} lies in the intervals 0–0.01, 0.01—0.09, and 0.09-1. Do this for both DES and AES, in both ECB and CBC mode. What would you expect to see if the ciphertexts were truly random strings? Explain the results that you find experimentally – do they indicate a weakness in any of the ciphers/encryption modes?

(d) Consider encrypting the following block of text:

> The following files are available: AA1, top secret; A4, top secret; B5, unclassified; Iraq.txt, top secret; DC, top secret; end.

(The text has exactly 128 ASCII characters.) What could an eavesdropping adversary learn if this text were encrypted using DES in ECB mode? What about CBC mode? (Feel free to encrypt it yourself to help answer this question.)

[2] Note that when checking a DES key, you should ignore the parity-check bits and only look at the random 56-bit portion of the key.