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

Problem Set 2

Due at *beginning* of class on Oct. 14

1. (**20 points.**) Show a chosen-ciphertext attack against the three modes of encryption (CBC, CFB, OFB) that are secure against chosen-plaintext attacks. In particular:

   (a) You are given a ciphertext $C = C_0, C_1, C_2, \ldots, C_n$ which is an encryption of some message $M = M_1, M_2, \ldots M_n$ using an unknown key k, where $|M_i| = |C_i|$ (and both are equal to the block length of the cipher), and you can assume that $n > 1$. You want to determine M, and are given no *a priori* information about it.

   (b) You have access to a *decryption oracle* $\mathcal{D}_k(\cdot)$ which will decrypt any ciphertext you give it, using the same key k that was used to encrypt C. *The only limitation is that you cannot ask it to decrypt C itself.* (Any other ciphertext is fair game.)

   (c) Describe in sufficient detail how you would go about recovering the message M. You can describe it in concise text or in pseudocode.

   (d) The TAs may set up a decryption oracle and give you "challenge" ciphertexts to try to decrypt. (See the HW2 FAQ page when it is up.) This is only intended for you to check your answer if you like, and you do not need to turn in any working program for this question.

2. (**10 points.**) In class, we mentioned that when using RSA it is sometimes advantageous to use "small" values of e (e.g., $e = 3$) to allow for efficient encryption. But in some cases (e.g., a server which is accepting — and decrypting — many messages simultaneously), it is more important to minimize the time required for *decryption*. A company has suggested achieving very efficient decryption by using "small" values of d (say, d prime and $d < 100$). Would you (as a consultant for this company) recommend such an approach? If yes, state (briefly) why. If no, give an explicit attack on such a system.

3. (**40 points.**) You are to implement the "basic" RSA encryption scheme. To be precise: this means that if the modulus N is, e.g., 1024 bits long, then a message m (with $|m| < 1024$) is encrypted by viewing m as an integer in $\mathbb{Z}_N^*$ and computing $m^e \bmod N$, where e is the public exponent. (We mentioned in class that this encryption scheme is not secure. Still, it is a useful starting point.)

   You should first implement a key generation program, which should operate as follows:

   - The program should take a command-line input, denoted ℓ, which determines the length of the primes p and q.
   - Your program should generate two, random ℓ-bit primes p and q. You should do this as discussed in class, by generating random ℓ-bit numbers and testing them

for primality. You should use the built-in Java routines for generating random numbers and for testing primality.

- Compute N, find the smallest valid value for e, and compute the appropriate value of d.
- Output the public key to a file pk.txt. The format should be as follows: the first line of this file should contain an integer which is equal to the length of N in binary. The second line should contain the modulus N, in hexadecimal. The third line should contain e in hexadecimal.
- Output the secret key to a file sk.txt. The format should be as follows: the first line of this file should contain the key-length ℓ (as an integer). The second line should contain the modulus N, in hexadecimal. The third line should contain d in hexadecimal.

You should also implement an encryption program, which should operate as follows:

- It should take two command-line arguments: the first specifies the file in which the public key is stored (the format will be as above), and the second specifies a file containing the message. The message will be in ASCII text.
- Your program should view each ASCII character as its 8-bit equivalent. *For the purposes of this problem, you may assume that the modulus in the public-key file will be 1024 bits long, and the message to be encrypted will contain at most 800 bits (i.e., 100 ASCII characters).* Include a README file in which you explan how you map such messages to $\mathbb{Z}_N^*$.
- Encrypt the message, and output the ciphertext to a file ciphertext.txt. The ciphertext should be in hexadecimal format.

Finally, you should implement a decryption program which reverses the above (i.e., takes a secret-key file and ciphertext file as input, and outputs the appropriate plaintext to a file in ASCII format). Make sure you also reverse whatever procedure you used during encryption to map messages to $\mathbb{Z}_N^*$.

4. (**30 points.**) Describe how you would modify the above program to allow for public-key encryption of arbitrary-length messages. (As above, you may assume that the message in ASCII text, so its length [in bits] will be a multiple of 8.) You should, at a minimum, address the following points:

   - How will you encrypt messages that are longer then the modulus?
   - How will you let the recipient know when the message ends, given that messages have variable length?

   Describe your modifications in sufficient detail for someone to be able to implement it. You do not need to implement it yourself. Your proposal will be graded based (in part) on its efficiency.