EE 8372 CRYPTOGRAPHY & DATA SECURITY

Homework 8 **Professor Dunham**
24 March 2020 **Due: 31 March 2020**

Review Text: Chapter 3, section 3.4; Chapter 7, sections 7.1-7.4 and 7.6; and Chapter 8, subsections 8.1.3 and 8.2.4.
Suggested Reading in Menezes, Oorschot and Vanstone: Chapter 2, section 4, 5, and 6; Chapter 3, sections1, 2, 3, 6, 7 and 8; Chapter 4, sections 1, 2, 3 and 4; Chapter 8, sections 1 and 2; and Chapter 9, sections 1, 2, 3 and 4.
Read Lightly FIPS 180-2, Secure Hash Standard (SHS), 2008 October 17.

1. Compute $7777^{45} \bmod (77773)$ by using repeated squaring (**show all steps**).

2. The ciphertext 9472 was obtained from the RSA algorithm using modulus $pn = 11413 = 101 \times 113$ and $e = 7467$. Find the plaintext.

3. In order to increase security, Bob chooses modulus pn and two encryption exponents e_1 and e_2 . He asks Alice to encrypt her message m to him by first computing $c_1 \equiv m^{e_1} \pmod{pn}$, then encrypting $c_2 \equiv c_1^{e_2} \pmod{pn}$. Alice then sends c_2 to Bob. Does this double encryption increase security over single encryption? Why or why not?

4. This problem looks at some aspects of prime numbers.

   (a) Let p be an n-bit number (the number has exactly n bits and the highest bit must be a 1). What are the minimum and maximum possible values of p?

   (b) What is a reasonable estimate for the number of n-bit prime numbers?

   (c) Approximately how many 1024-bit numbers are prime? *Hint:* This is a LARGE number, use scientific notation to represent your answer.

   (d) The current NIST standard (NIST 800-131A Revision 1) recommendation for the modulus (pn) of a RSA algorithm is that it be a minimum of 2048-bits. Assuming that p is a 1024-bit prime number, what is the probability than an odd 1024-bit random number is prime?

   (e) Is an exhaustive search attack feasible for finding a 1024-bit prime number used in a 2048-bit or larger RSA modulus?

5. Using the Miller-Rabin random primality testing algorithm, determine which of the following hex numbers are prime with error probability less than 10^{-8}. All the numbers in the problem are 30-bit or smaller numbers. You can program the Miller-Rabin algorithm in your favorite programming language using 64-bit integers and use integer arithmetic operations and modulo reductions. *Hint:* It is suggested that you use the UNIX program bc (a C style interactive language) as well as the primality-testing.bc software package. Information on the program bc as well as several programming examples are available on the course web site.

   (a) 0x108C1
   (b) 0x6CE45
   (c) 0x17C79D
   (d) 0x1B206B
   (e) 0xA98AC6D
   (f) 0x7CAB1C23

   Optional: If you are interested in testing some larger numbers, use your Miller-Rabin random primality testing algorithm to determine if the following numbers are composite or tests prime with an error probability of less than 10^{-8}.

   (a) 0x97340264E77E568659
   (b) 0x5AD789FB37AADE289F
   (c) 0x7D25FB921140550EBFB0D84B
   (d) 0x34BA3709DF728A732FDB1787
   (e) 0x62A6F3AFC5E548D4EC00DEB0873FDE7386C10994190130A7
   (f) 0xD37ADA29632ADDE177CE24E4671D5F59DEDB7E193EA8B117

   Comments: Both the Miller-Rabin and the Solovay-Strassen random primality testing algorithms can fail for a set of composite numbers called pseudoprimes – composite numbers that pass all test as a prime number. While this is an unlikely event, it can occur. It is generally recommend that a fast primality testing algorithm such as the Miller-Rabin or Solovay-Strassen be used to identify a prime candidate and then use a different class of random primality testing algorithms such as the Lucas primality test described in the NIST standard FIPS 186-4 to provide add additional confidence in the primality of a prime candidate. Finally, since determining primality is in the class *P*, one could use one of the deterministic primality algorithms to make a final confirmation.