Assignment 2
Gunnar Yonker

1.

(1) 9^9602 mod 65 = 16

p = 65 and a = 9

9^(65-1) mod 65 = 1

9^9602 mod 65 = (9^64)^150 mod 65 * 9^2 mod 65

= 1^150 * 81 mod 65

= 16 mod 65

= 16

(2) 6^324 mod 17 = 4

p = 17 and a = 6

6^324 mod 17 = (6^16)^20 mod 17 * 6^4 mod 17

= (16)^20 mod 17 * (1296 mod 17)

= 4^20 mod 17 *4

= 16 mod 17

= 4

2.

(1)

n = 5*11 = 55

phi(n) = (5-1)(11-1) = 40

k =1

27 * d = 40 * 1 + 1

27 * d = 41

d = 41/27 = 3

Private key d = 3

Encrypt with M=31

C = Me (mod n) = 31 ^27 (mod 55) = 47

Decrypt C=47

M = Cd(mod n) = 47^3 (mod 55) = 31

Assignment 2
Gunnar Yonker

(2)

n = 7 * 13 = 91

phi(n) = (7-1)(31-1) = 72

k = 1

5 * d = 71 * 1 + 1

5 * d = 73

d = 73/5 = 14.6

k = 2

5 * d = 72 * 2 + 1

5d = 145

d = 29

Private key d = 29

Encrypt with M =15

C = Me (mod n) = 15^5 (mod 91) = 51

Decrypt C = 51

M = Cd (mod n) = 51^29 (mod 91) = 15

3.

(1)

The RSA algorithm uses a public key and a private key cryptography system to ensure that the communication is secure. The RSA decryption process uses the ciphertext C and private key (d), and recover the original plaintext M using the following operation:
Encryption:

M = C^d mod n

Decryption:

C = M^d mod n

In this equation n is the modulus and the private key as stated above is d. This process is the equivalent of finding the remainder when C^d is divided by n.

Now the issue of the plaintext needing to be less than n. The value of M needs to be less than n because if it was greater than or equal to n, this would result in the plaintext M being some other value that is equivalent to M. The decryption process would be incorrect. When the operation takes place the result

of a mod b should always be less than b, similarly, the decrypted value of the mod n operation means that the plaintext M value will always be less than n. In order to get the original message the value needs to be less than n for the input otherwise you will end up with the incorrect plaintext during the decryption compared to the originally encrypted plaintext.

(2)

The first step to encrypt an M that is 5,000 digits long when the n is about 400 digits long would be to split the M into multiple blocks that are less than the value of n. Each of the blocks would then be encrypted individually using the RSA encryption, which would result in multiple ciphertexts that would then be decrypted individually, reassembled, and then you would have the original plaintext message. The receiver B would receive each of the block pieces from A that were encrypted using B's private key. B would then decrypt each individual block piece using the private key and decryption process, then assemble the blocks to have the full completed plaintext.

4.

(1)

I think it is difficult to know if this is a good pseudo random number generator based on one generation of bits. A good pseudo random number generator could require statistical randomness to show that the generated numbers are truly random numbers, amongst other qualities such as the numbers being independent of each other. Based on this singular sequence of bits in my opinion I can consider it a random number because it does not have any correlation to the assignment itself that I can see a pattern from or any other reason to assume it is not random. However, in my opinion I don't think that I can consider it a good pseudo random number generator based off of this one sequence of bits, because the next sequence of bits could only increase by 1 or be the same "random number". So without more data or additional information about the generator itself I don't think that I could say that it is a good random number generator based off of the sequence of bits given.

(2)

plaintext(hex) : 3x 2f ab e4

key: 00001111000110000100001001000010

ciphertext(hex): 3f e9 bb 74

(3)

plaintext(hex): 25 f8 57 cc

key: 00001111000110000100001001000010

ciphertext(hex): 26 3e 47 5c

Assignment 2
Gunnar Yonker

5.

(1)

A pseudo random number generator (PRNG) is a generator that produces numbers that are statistically random but not truly random unlike a true random number generator (TRNG). PRNGs have a few advantages over TRNGs. For debugging purposes PRNGs have the advantage of reproducibility because you can produce the same sequence of numbers using the same seed with a PRNG, unlike a TRNG which cannot be reproduced or predicted. Another advantage is that PRNGs are usually quite faster than a TRNG because TRNGs have to use algorithms to create their numbers which takes more time. Another aspect of PRNGs that can be an advantage or disadvantage depending on how you look at it is that PRNGs have a level of predictability since they are statistically random, so a certain degree of predictability can be expected with PRNGs. Typically, PRNGs are easier to use when compared to TRNGs.

(2)

I think that I would most likely choose to use the Linear Feedback Register Sequence Generator because it is a simple and fast key generator for stream ciphers. They are able to generate a long key in the form of long sequences that appear randomly generated which makes it suitable for stream ciphers. Also since it is simple and fast, that makes it easy to implement. The Blum Blum Shub generator is slower than the LFSR generator and requires a lot more resources to operate in comparison. For a stream cipher speed and efficiency are important which is why I did not pick this generator. RC 4 bit generator was very widely used which lead to certain vulnerabilities and weaknesses being discovered leading to it being a less secure generator. It is important that they generated keys are very secure as that is the point of the encryption of data which is why I did not pick the RC 4 bit generator due to the known vulnerabilities which could lead to security issues.