CPE-321 Introduction of Computer Security
Assignment: Public Key Cryptography Implementation

(Revised Hartman 1/22/23 to align with lecture terminology)

Objectives

The objectives for this lab assignment are as follows:

- To explore public key cryptography
    - Implementing the Diffie-Hellman Key Exchange protocol
    - Implementing RSA encryption scheme

Background

In this lab, you will explore asymmetric key (public key) cryptography by implementing the Diffie-Hellman Key Exchange protocol and RSA encryption scheme. You will explore the properties of these schemes, and see **how naive implementations can lead to insecurity**. Python and PyCryptodome (or some other high-level language with decent crypto support) are recommended to use for this lab.

Tasks:

Please read the following tasks carefully, and finish the lab assignment accordingly:

Task 1: Implement Diffie-Hellman Key Exchange: The goal of this task is to get your public key crypto juices flowing by implementing one of the single most important discoveries in modern cryptography: Diffie Hellman Key Exchange. In a single program, emulate the following protocol:

Alice		Bob	
Privately Computes	**Sends**	**Privately Computes**	**Sends**
	q, α		
Pick random element $X_A \in \mathbb{Z}_q$ $Y_A = \alpha^{(X_A)} \bmod q$		Pick random element $X_B \in \mathbb{Z}_q$ $Y_B = \alpha^{(X_B)} \bmod q$	
	Y_A		Y_B
$s = (Y_B)^{(X_A)} \bmod q$		$s = (Y_A)^{(X_B)} \bmod q$	
$k = SHA256(s)$		$k = SHA256(s)$	
m_0 = “Hi Bob!” $c_0 = AES\text{-}CBC_k(m_0)$	c_0	m_1 = “Hi Alice!” $c_1 = AES\text{-}CBC_k(m_1)$	c_1

Try it in a small group first, setting $q=37$ and $\alpha=5$. Confirm Alice and Bob compute the same symmetric key k. Truncate the output of SHA256 to 16 bytes, so that you can use it to AES-CBC encrypt some messages between Alice and Bob.

Now, let’s use some “real life” numbers. IETF suggests the following 1024-bit parameters:

q =

B10B8F96 A080E01D DE92DE5E AE5D54EC 52C99FBC FB06A3C6
9A6A9DCA 52D23B61 6073E286 75A23D18 9838EF1E 2EE652C0
13ECB4AE A9061123 24975C3C D49B83BF ACCBDD7D 90C4BD70
98488E9C 219A7372 4EFFD6FA E5644738 FAA31A4F F55BCCC0
A151AF5F 0DC8B4BD 45BF37DF 365C1A65 E68CFDA7 6D4DA708
DF1FB2BC 2E4A4371

α =

A4D1CBD5 C3FD3412 6765A442 EFB99905 F8104DD2 58AC507F

D6406CFF 14266D31 266FEA1E 5C41564B 777E690F 5504F213 160217B4 B01B886A 5E91547F 9E2749F4 D7FBD7D3 B9A92EE1 909D0D22 63F80A76 A6A24C08 7A091F53 1DBF0A01 69B6A28A D662A4D1 8E73AFA3 2D779D59 18D08BC8 858F4DCE F97C2A24 855E6EEB 22B3B2E5

Modify your implementation to use these parameters, make sure Alice and Bob can compute the same symmetric key k, and correctly exchange some encrypted messages.

Task 2: Implement MITM key fixing & negotiated groups: Diffie-Hellman is only secure against a passive (eavesdropping) adversary. Very bad things can happen if the adversary is able to tamper with the numbers in the protocol.

1). Modify your implementation from Task 1 in the following way:

Alice		**Mallory**	**Bob**	
Computes	**Sends**	**Modifies**	**Computes**	**Sends**
	q, α			
$X_A \in \mathbb{Z}_q$ $Y_A = \alpha^{(X_A)} \bmod q$			$X_B \in \mathbb{Z}_q$ $Y_B = \alpha^{(X_B)} \bmod q$	
	Y_A	$Y_{A \to q}$		
		$Y_{B \to q}$		Y_B
$s = (Y_B)^{(X_A)} \bmod q$			$s = (Y_A)^{(X_B)} \bmod q$	
k = SHA256(s)			k = SHA256(s)	
m_0 = “Hi Bob!” c_0 = AES-CBC$_k$(m_0)	c_0		m_1 = “Hi Alice!” c_1 = AES-CBC$_k$(m_1)	c_1

2). Repeat this attack, but instead of tampering with Y_A and Y_B, tamper with the generator α. Show that Mallory can recover the messages m_0 and m_1 from their ciphertexts by setting α to 1, q, or q-1.

Task 3: Implement "textbook" RSA & MITM Key Fixing via Malleability:

1). RSA has two core components: key generation and encryption/decryption. (90% of the work is in implementing key generation.) Your implementation should support variable length primes (up to 2048 bits), and use the value e=65537. Feel free to use your cryptographic library's interface for generating large primes, but implement the rest-including computing the multiplicative inverse - yourself.

Encrypt and decrypt a few messages to yourself to make sure it works. Remember messages must be integers in Z_n^* (that is, less than n, the product of the two primes). You can convert an ASCII string to hex, and then turn that hex value into an integer.

2). From 1). you just implemented "textbook" RSA, and it is widely insecure. Because it is too slow and inconvenient to operate on a large amount data directly, RSA is often used to exchange a symmetric key that will be used to encrypt future messages. It would be terrible, of course, if an adversary were able to learn that key. And, that's what we're about to do. One of textbook RSA's great weaknesses is its malleability, i.e. an active attacker can change the meaning of the plaintext message by performing an operation on the respective ciphertext. To demonstrate the dangers of malleability, implement the following protocol:

Alice		Mallory	Bob	
Computes	**Sends**	**Modifies**	**Computes**	**Sends**
	n,e			
			$s \in \mathbb{Z}^*_n$ $c = s^e \bmod n$	
				c
		c′ = F(c)		
$s = c'^d \bmod n$				
k = SHA256(s)				
m = "Hi Bob!" $c_0 = \text{AES-CBC}_k(m)$	c_0			

Find the operation F() that Mallory needs to apply to the ciphertext c that will allow her to decrypt the ciphertext c_0. Hint: Mallory knows Alice's public key (n,e) and can encrypt her own messages to Alice to allow Mallory to know the value of s.

Give another example of how RSA's malleability could be used to exploit a system (e.g. to cause confusion, disruption, or violate integrity).

Malleability can also affect signature schemes based on RSA. Consider the scheme: Sign(m,d) = $m^d \bmod n$

Suppose Mallory sees the signatures for two messages m_1 *and* m_2. Show how Mallory can create a valid signature for a third message, $m_3 = m_1 \cdot m_2$.

Alice		Bob	
Privately Computes	**Sends**	**Privately Computes**	**Sends**
	$q,\ \alpha$		
$X_A \in \mathbb{Z}_q$ $Y_A = \alpha^{(X_A)} \bmod q$		$X_B \in \mathbb{Z}_q$ $Y_B = \alpha^{(X_B)} \bmod q$	
	Y_A		Y_B
$s = (Y_B)^{(X_A)} \bmod q$ Forget X_A		$s = (Y_A)^{(X_B)} \bmod q$ Forget X_B	
k = SHA256(s)		k = SHA256(s)	
m_0 = "Hi Bob!" $c_0 = \text{AES-CBC}_k(m_0)$	c_0	m_1 = "Hi Alice!" $c_1 = \text{AES-CBC}_k(m_1)$	c_1

Protocol 1: A Diffie-Hellman Key-agreement

Alice		Bob	
Privately Computes	**Sends**	**Privately Computes**	**Sends**
RSA key pair $<A_{pub}, A_{pri}>$			
	A_{pub}		
		Pick random element $x \in \mathbb{Z}^*_n$ $y = \text{RSA}(A_{pub}, x)$	
			y
$x = \text{RSA}^{-1}(A_{pri}, y)$ k = SHA256(x)		k = SHA256(x)	
m_0 = "Hi Bob!" $c_0 = \text{AES-CBC}_k(m_0)$	c_0	m_1 = "Hi Alice!" $c_1 = \text{AES-CBC}_k(m_1)$	c_1

Protocol 2: An RSA Key Exchange

Questions:

1. For task 1, how hard would it be for an adversary to solve the Diffie Hellman Problem (DHP) given these parameters? What strategy might the adversary take?
2. For task 1, would the same strategy used for the tiny parameters work for the large values of q and α? Why or why not?
3. For task 2, why were these attacks possible? What is necessary to prevent it?
4. For task 3 part 1, while it's very common for many people to use the same value for e in their key (common values are 3, 7, 216+1), it is very bad if two people use the same RSA modulus n. Briefly describe why this is, and what the ramifications are.

Submission: Submit a report describing what you did and what you observed. Include any code that you wrote, as well as answers to any questions (there are in italics). Please include any explanations of the surprising or interesting observations you made.

Write at a level that demonstrates your technical understanding, and do not shorthand ideas under the assumption that the reader already "knows what you mean". Think of writing as if the audience was a smart colleague who may not have taken this class.

Describe what you did in sufficient detail that it can be reproduced. Please do not use screenshots of the VM to show the commands and code you used, but rather

paste (or carefully retype) these into your report from your terminal. Submit your completed write up to Canvas in PDF format.