Public Ciphers

Caleb Dennis

CPE321-03

## Task 1: Implement Diffie-Helman key exchange

The Diffie-Helman key exchange allows users to create a shared secret without communicating any information that would make the secret insecure. In task 1 Alice and Bob independently compute the same shared secret which can be used as a key to securely exchange information. To find the shared secret an adversary must find the prime number that Alice or Bob each chose. To find this number the adversary needs to compute the log base g of Alice’s public key. An adversary could check all primes, but if Alice has chosen a reasonably large prime this process takes a lot of computing.

from Crypto.Util.number import getPrime

from Crypto.Util.Padding import pad, unpad

from Crypto.Random import get\_random\_bytes

from Crypto.Hash import SHA256

from Crypto.Cipher import AES

from Crypto.Cipher import AES

def main():

    p = 37

    g = 5

    Alice\_prime = getPrime(16)

    Bob\_prime = getPrime(16)

    print(f"Alice's prime: {Alice\_prime}")

    print(f"Bob's prime: {Bob\_prime}")

    Alice\_public = pow(g, Alice\_prime) % p

    Bob\_public = pow(g, Bob\_prime) % p

    Alice\_shared\_secret = bytes(pow(Bob\_public, Alice\_prime) % p)

    Bob\_shared\_secret = bytes(pow(Alice\_public, Bob\_prime) % p)

    Alice\_hasher = SHA256.new()

    Alice\_hasher.update(Alice\_shared\_secret)

    alice\_key = Alice\_hasher.digest()

    Bob\_hasher = SHA256.new()

    Bob\_hasher.update(Bob\_shared\_secret)

    bob\_key = Bob\_hasher.digest()

    iv = get\_random\_bytes(16)

    print(f"Bob's key {alice\_key}\nAlice's key {bob\_key}\n")

    Alice\_msg = pad(bytes("Hi Bob, i'm Alice", "ascii"), 16)

    print(f"Alice: {unpad(Alice\_msg, 16)}")

    Alice\_encrypter = AES.new(alice\_key, AES.MODE\_CBC, *iv*=iv)

    Alice\_sends\_to\_Bob = Alice\_encrypter.encrypt(Alice\_msg)

    Bob\_encrypter = AES.new(bob\_key, AES.MODE\_CBC, *iv*=iv)

    Bob\_received\_from\_Alice = Bob\_encrypter.decrypt(Alice\_sends\_to\_Bob)

    print(f"Bob received: {unpad(Bob\_received\_from\_Alice, 16)}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

## Task2:

The Diffie-Helman key exchange can be unsecure if an adversary is able to adjust the numbers in the protocol. As shown in task2, an adversary can gain access to both the public and private keys of Bob and Allice by manipulating the numbers in the protocol so that the math to generate a private key returns a specific value.

The first example of this can be observed in main1. In this scenario, the adversary adjusts Alice and Bob’s public key by setting them to be the value of ‘p’. In doing this, the adversary is able to predict that the private key of both Alice and Bob will be 0 since p^prime\_val % p will always equate to be 0 for any prime value.

The second example can be observed in main2. In this scenario, the adversary adjusts the ‘g’ value to be either 1, p, or p-1.

* In the case of g = 1, Alice and Bob’s public and private key will always be 1
  + Public: 1^prime\_val % p will always equal 1
  + Private: 1^prime\_val % p will always equal 1
* In the case of g = p, Alice and Bob’s public and private key will always be zero
  + Public: p^prime\_val % p will always equal 0
  + Private: 0^prime\_val % p will always equal 0
* In the case of g = (p-1), Alice and Bob’s public and private will always be (p-1)
  + Public: (p-1)^prime\_val % p will always equal (p-1) as long as the exponent is odd (in this case since it’s prime then it is always odd so this works)
  + Private: (p-1)^prime\_val % p will always equal (p-1)

Code:

from Cryptodome.Util.number import getPrime

from Cryptodome.Util.Padding import pad, unpad

from Cryptodome.Random import get\_random\_bytes

from Cryptodome.Hash import SHA256

from Cryptodome.Cipher import AES

# Part 1

def main\_1():

p = 37

g = 5

# Get prime for Alice and Bob

Alice\_prime = getPrime(16)

Bob\_prime = getPrime(16)

print(f"Alice's prime: {Alice\_prime}")

print(f"Bob's prime: {Bob\_prime}")

Alice\_public = pow(g, Alice\_prime) % p

Bob\_public = pow(g, Bob\_prime) % p

# Mallory's modifications

Alice\_public = p

Bob\_public = p

# Because Mallory changed public to 'p', it will always mod to 0 no matter the prime

Alice\_shared\_secret = bytes(pow(Bob\_public, Alice\_prime) % p)

Bob\_shared\_secret = bytes(pow(Alice\_public, Bob\_prime) % p)

# Mallory can now be sneaky

Mallory\_stolen\_secret = bytes(0)

Mallory\_hasher = SHA256.new()

Mallory\_hasher.update(Mallory\_stolen\_secret)

mallory\_key = Mallory\_hasher.digest()

Alice\_hasher = SHA256.new()

Alice\_hasher.update(Alice\_shared\_secret)

alice\_key = Alice\_hasher.digest()

Bob\_hasher = SHA256.new()

Bob\_hasher.update(Bob\_shared\_secret)

bob\_key = Bob\_hasher.digest()

# I think the IV is also just the key in this case

# Cuz if not, not quite sure how mallory gets this value IRL

iv = get\_random\_bytes(16)

print(f"Bob's key {alice\_key}\nAlice's key {bob\_key}\n")

Alice\_msg = pad(bytes("Hi Bob, i'm Alice", "ascii"), 16)

print(f"Alice: {unpad(Alice\_msg, 16)}")

Alice\_encrypter = AES.new(alice\_key, AES.MODE\_CBC, iv) #using alice\_key for iv

Alice\_sends\_to\_Bob = Alice\_encrypter.encrypt(Alice\_msg)

Bob\_encrypter = AES.new(bob\_key, AES.MODE\_CBC, iv) #using bob\_key for iv

Bob\_received\_from\_Alice = Bob\_encrypter.decrypt(Alice\_sends\_to\_Bob)

print(f"Bob received: {unpad(Bob\_received\_from\_Alice, 16)}")

# Mallory swooping in

Mallory\_encryptor = AES.new(mallory\_key, AES.MODE\_CBC, iv)

Mallory\_stolen\_from\_Alice = Mallory\_encryptor.decrypt(Alice\_sends\_to\_Bob)

print(f"Mallory Intercepted: {unpad(Mallory\_stolen\_from\_Alice, 16)}")

if \_\_name\_\_ == "\_\_main\_\_":

main\_1()

def main\_2():

p = 37

g = 5

# Get prime for Alice and Bob

Alice\_prime = getPrime(16)

Bob\_prime = getPrime(16)

print(f"Alice's prime: {Alice\_prime}")

print(f"Bob's prime: {Bob\_prime}")

# Mallory's Modifications

# g = 1 # This means the public will be 1 & secret will be 1

# Mallory\_stolen\_secret = bytes(1)

# g = p # This means the public and secret will be 0

# Mallory\_stolen\_secret = bytes(0)

g = p - 1 # This means the public and secret will be p - 1

Mallory\_stolen\_secret = bytes(p-1) # Ex) (p = 7, g = p-1 = 6, prime=3) => 6^3 % 7 = 6 = (p-1)

Alice\_public = pow(g, Alice\_prime) % p

Bob\_public = pow(g, Bob\_prime) % p

Alice\_shared\_secret = bytes(pow(Bob\_public, Alice\_prime) % p)

Bob\_shared\_secret = bytes(pow(Alice\_public, Bob\_prime) % p)

# Mallory can now be sneaky

Mallory\_hasher = SHA256.new()

Mallory\_hasher.update(Mallory\_stolen\_secret)

mallory\_key = Mallory\_hasher.digest()

Alice\_hasher = SHA256.new()

Alice\_hasher.update(Alice\_shared\_secret)

alice\_key = Alice\_hasher.digest()

Bob\_hasher = SHA256.new()

Bob\_hasher.update(Bob\_shared\_secret)

bob\_key = Bob\_hasher.digest()

# I think the IV is also just the key in this case

# Cuz if not, not quite sure how mallory gets this value IRL

iv = get\_random\_bytes(16)

print(f"Bob's key {alice\_key}\nAlice's key {bob\_key}\n")

Alice\_msg = pad(bytes("Hi Bob, i'm Alice", "ascii"), 16)

print(f"Alice: {unpad(Alice\_msg, 16)}")

Alice\_encrypter = AES.new(alice\_key, AES.MODE\_CBC, iv) #using alice\_key for iv

Alice\_sends\_to\_Bob = Alice\_encrypter.encrypt(Alice\_msg)

Bob\_encrypter = AES.new(bob\_key, AES.MODE\_CBC, iv) #using bob\_key for iv

Bob\_received\_from\_Alice = Bob\_encrypter.decrypt(Alice\_sends\_to\_Bob)

print(f"Bob received: {unpad(Bob\_received\_from\_Alice, 16)}")

# Mallory swooping in

Mallory\_encryptor = AES.new(mallory\_key, AES.MODE\_CBC, iv)

Mallory\_stolen\_from\_Alice = Mallory\_encryptor.decrypt(Alice\_sends\_to\_Bob)

print(f"Mallory intercepted: {unpad(Mallory\_stolen\_from\_Alice, 16)}")

if \_\_name\_\_ == "\_\_main\_\_":

main\_2()

**Task 2 Output:**

As we can see based on the output, Mallory was able to intercept the message sent to Bob by manipulating either Alice and Bob’s public key or by manipulating the ‘g’ value.

Alice's prime: 63647

Bob's prime: 49603

Bob's key b"\xe3\xb0\xc4B\x98\xfc\x1c\x14\x9a\xfb\xf4\xc8\x99o\xb9$'\xaeA\xe4d\x9b\x93L\xa4\x95\x99\x1bxR\xb8U"

Alice's key b"\xe3\xb0\xc4B\x98\xfc\x1c\x14\x9a\xfb\xf4\xc8\x99o\xb9$'\xaeA\xe4d\x9b\x93L\xa4\x95\x99\x1bxR\xb8U"

Alice: b"Hi Bob, i'm Alice"

Bob received: b"Hi Bob, i'm Alice"

Mallory Intercepted: b"Hi Bob, i'm Alice"

Alice's prime: 50051

Bob's prime: 63367

Bob's key b'm\xb6\_\xd5\x9f\xd3V\xf6r\x91@W\x1b[\xcdk\xb3\xb84\x92\xa1n\x1b\xf0\xa3\x88DB\xfc<\x8a\x0e'

Alice's key b'm\xb6\_\xd5\x9f\xd3V\xf6r\x91@W\x1b[\xcdk\xb3\xb84\x92\xa1n\x1b\xf0\xa3\x88DB\xfc<\x8a\x0e'

Alice: b"Hi Bob, i'm Alice"

Bob received: b"Hi Bob, i'm Alice"

Mallory intercepted: b"Hi Bob, i'm Alice"