HW 2
James Foti
Red ID: 820124143
CS 574 - Computer Security
Professor Song
3/11/2020

QUESTION 1) In this problem we will compare the security services that are provided by digital signatures (DS) and message authentication codes (MAC). We assume that Oscar is able to observe all messages sent from Alice to Bob and vice versa. Oscar has no knowledge of any keys but the public one in case of DS. State whether and how (i) DS and (ii) MAC protect against each attack. The value auth(x) is computed with a DS or a MAC algorithm, respectively.

Part (a): (Message integrity) Alice sends a message x = “Transfer $1000 to Mark” in the clear and also sends auth(x) to Bob. Oscar intercepts the message and replaces “Mark” with “Oscar.” Will Bob detect this?
Part (a) Solution: DS and MAC will both detect this integrity attack.

Part (b): (Replay) Alice sends a message x = “Transfer $1000 to Oscar” in the clear and also sends auth(x) to Bob. Oscar observes the message and signature and sends them 100 times to Bob. Will Bob detect this?
Part (b) Solution: Sending the same message again will not be detected by DS or MAC.

Part (c): (Sender authentication with cheating third party) Oscar claims that he sent some message x with a valid auth(x) to Bob but Alice claims the same. Can Bob clear the question in either case?
Part (c) Solution: Bob can verify the authorized sender by using DS and MAC.

Part (d): (Authentication with Bob cheating) Bob claims that he received a message x with a valid signature auth(x) from Alice (e.g., “Transfer $1000 from Alice to Bob”) but Alice claims she has never sent it. Can Alice clear this question in either case?
Part (d) Solution: Alice can clear this up if the message was sent by using DS only.

QUESTION 2) In this question,
Part (a): Consider the following hash function. Messages are in the form of a sequence of decimal numbers, $M = (a_1, a_2, \ldots, a_t)$. The hash value h is calculated as $(\sum_{i=1}^{t} a_i) mod\ n$,for some predefined value n. Does this hash function satisfy the basic properties of a hash function? Explain your answer.

Part (a) Solution: $(\sum_{i=1}^{t} a_i) mod\ n$, where n = predefined value
This function does not satisfy all of the basic properties of a hash function because:

- It is feasible to find pairs that have the same hash value meaning that it is feasible to find a pair (x, y) where $H(y) = H(x)$.
- For any given code block x, it is feasible to find $y \neq x$ where $H(y) = H(x)$.

Part (b): Repeat part (a) for the hash function $h = \sum_{i=1}^{t} a_i^2$.

Part (b) Solution
This function does not satisfy all of the basic properties of a hash function because:

- It is feasible to find pairs that have the same hash value meaning that it is feasible to find a pair (x, y) where $H(y) = H(x)$.
- For any given code block x, it is feasible to find $y \neq x$ where $H(y) = H(x)$.

Part (c): Calculate the hash function of part (b) for $M = [189, 632, 900, 722, 349]$ and $n = 989$

Part (c) Solution:

$$h = \sum_{i=1}^{t} a_i^2 \; mod \; n$$
$$h = (189^2 + 632^2 + 900^2 + 722^2 + 349^2) \; mod \; 989$$
$$= 1,888,230 \; mod \; 989$$
$$= 229$$

QUESTION 3) Considering the concept of salted passwords, answer the following questions:

Part (a): Bob thinks that generating and storing a random salt value for each user id is a waste. Instead, he is proposing that his system administrators use a cryptographic hash of the user id as its salt. Describe whether this choice impacts the security of salted passwords and include an analysis of the respective search spaces.

Part (a) Solution: Let's say that the length of a salt $= 64$ bits. If Bob uses salted passwords, the search spaces will increase by 2^{64} times. If Bob uses a cryptographic hash of the user id's as their salts, the search space will be reduced greatly and will make attacks more feasible because if more than one user id's password is the same, then they will map to the same hash value which will cause a security risk. It is safer to store a random salt value for each user id because if they two user id's have the same password, their salt value will still be different and thus the string to be hashed and the hashed value (password + salt value) will be different and an adversary will find it harder to map a password with its correct hashed value.

Part (b): The attacker wants to minimize the number of bytes that she has to hash to conduct a dictionary attack. How can she take advantage of the structure of our hash to minimize the number of bytes she must hash to compute the combination of every dictionary word with every possible salt?

- i. Would the same attack be more, less, or just as effective if you stored h(salt | password) in the password file? Why or why not?
- ii. Would the same attack be more, less, or just as effective if you stored h(salt | password | salt) in the password file? Why or why not?

Part (b) Solution: It would not make a difference either way because there are just to many possibilities of salt values and hash values for the attacker. Dictionary attacks become more difficult to implement becasue the attacker cannot practically precompute the hashes.

QUESTION 4)
Part (a): What is meant by the strong collision resistance property of a hash function?

Part (a) Solution: This means that it is very hard (or not possible) to find two inputs that hash to the same output. More formally: tt is not feasible to find a pair $x \neq x'$ such that $h(x) = h(x')$

Part (b): Suppose $H(m)$ is a collision-resistant hash function that maps a message of arbitrary bit length into an n-bit hash value. Is it true that, for all messages x, x' with $x \neq x'$, we have $H(x) \neq H(x')$? Explain your answer.

Part (b) Solution: Because $H(m)$ is collision-resistant, this means that it is infeasible to find the value x' where $x' \neq x$ with $H(x') = H(x)$. This means that a different message cannot be found using the same hash value and proves that the statement above is false for all unique messages of x and x'.

QUESTION 5)
Part (a): HMAC can be used to simultaneously verify both data integrity and authenticity of the message:

- i. Briefly explain how data integrity is verified
- ii. Briefly explain what value is used to verify authenticity of the message

Part (a) Solution:

- i. By comparing the received HMAC hash with the calculated HMAC hash using the received message m and the secret key K. If both values are the same, then the message has not been corrupted.
- ii. The authenticity is verified by comparing the received HMAC hash with the calculated HMAC hash using the secret key and the received message.

Part (b): Now Let's say Alice has a long sentence, like

"The quick brown fox jumped over the lazy dog."

She needs to keep this string encrypted, so Alice uses an HMAC. Alice wants to do prefix searches for this string, so assume that she also stores all possible prefixes of this string, like HMAC("T"), HMAC("Th"), HMAC("The"), HMAC("The "), etc.

In the above scenario, would storing these set of HMAC values, make things unsecure? Could an adversary work out the key, or any part of the plaintext, given a series of HMAC values for all of a given input's prefix values? Assume that the instance of HMAC is a secure MAC function.

Part (b) Solution:
Because we are assuming that the instance of HMAC is a secure MAC function, an adversary would not be able to recover the key or any part of the the plaintext. It would not be efficient or practical even if the adversary has access to many MACs and their corresponding plaintexts.

QUESTION 6) Write a program to implement the RSA algorithm. The program should read the data from a text file, encrypt the message using the keys and write it back to a separate text file. The values of p and q should be provided by the user during runtime. Your program should be able to select appropriate vale of e based on the respective calculations. Your program should also be able to decrypt the encrypted message.

Function Description: Your program should have two separate functions for encryption and decryption each of which will accept file reference and respective keys as function arguments. The functions should encrypt/decrypt to the text files as mentioned above and should not return anything to the main program.

Note: You should write separate functions for primary calculations and for value checks. You can write the program in C++ / Java / Python. You are supposed to write your own functions. No in built functions and third-party APIs will be allowed.

Question 6 Solution:

Solution References

- The RSA Encryption Algorithm (1 of 2: Computing an Example) - https://www.youtube.com/watch?v=4zahvcJ9glg (https://www.youtube.com/watch?v=4zahvcJ9glg)
- The RSA Encryption Algorithm (2 of 2: Generating the Keys) - https://www.youtube.com/watch?v=oOcTVTpUsPQ&t=627s (https://www.youtube.com/watch?v=oOcTVTpUsPQ&t=627s)

Note: This implementation does not check for errors from user-input and errors from text file references. This program is used just for educational purposes.

In [1]:
```
import random
```

In [2]:
```
def generate_lock_and_key():
    """
    The steps for the RSA algorithm are outlined nicely in the links above!
    """

    # NOTE: for this to work, 'p' and 'q' must be greater than 10. I have not figured ou
t why this is the case but this
    # is something I will continue to research on down the road.
    # For an example: p = 2 and q = 7 WILL NOT WORK but p = 11 and q = 13 WILL WORK!

    p = int(input("Enter a number for 'p' (p > 10): ")) # Step 1, Example: p = 11
    q = int(input("Enter a number for 'q' (q > 10): ")) # Step 1, Example: q = 13

    n = p*q # Step 2

    phi = (p - 1)*(q - 1) # Step 3

    e = choose_e(phi, n) # Step 4
    public_key = [e, n]

    d = mod_inverse(e, phi) # Step 5
    private_key = [d, n]

    return public_key, private_key
```

In [3]:
```
def choose_e(phi, n):
    e_interval = list(range(2, phi))

    for i in range(len(e_interval)):
        if is_coprime(e_interval[i], n) and is_coprime(e_interval[i], phi):
            return e_interval[i]
    return 1
```

In [4]:
```
def mod_inverse(a, m):
    """
    SOURCE: https://www.geeksforgeeks.org/multiplicative-inverse-under-modulo-m/
    """
    a = a % m;
    for x in range(1, m) :
        if ((a * x) % m == 1) :
            return x
    return 1
```

In [5]:
```
def gcd(a,b):
    """
    SOURCE: https://www.geeksforgeeks.org/python-math-gcd-function/
    """
    if a == 0:
        return b
    return gcd(b % a, a)
```

In [6]:
```
def is_coprime(a, b):
    """
    SOURCE: https://stackoverflow.com/questions/39678984/efficiently-check-if-two-number
s-are-co-primes-relatively-primes
    """
    return gcd(a, b) == 1
```

In [7]:
```
class RSA:
    def encrypt(self, file_reference, public_key):
        # Note: the txt file must be in the same local directory as this notebook file!
        with open(file_reference, 'r+', encoding="utf-8") as file:
            file_contents = file.read()

            file_contents = [ord(c) for c in file_contents] # Convert message into list
 of numbers based of ASCII values
            cipher_text = []

            for i in range(len(file_contents)):
                cipher = (file_contents[i]**public_key[0]) % public_key[1]
                cipher_text.append(cipher)

            cipher_text = [chr(num) for num in cipher_text] #Convert list of ASCII RSA e
ncrypted values to a list of strings
            encrypted_data = "".join(cipher_text)

            # Clear txt file and upload the encrypted message back to the txt file
            file.seek(0)
            file.truncate()
            file.write(encrypted_data)

        return encrypted_data # I returned the data just so that you can see it without
 opening the txt file.

    def decrypt(self, file_reference, private_key):
        # Note: the txt file must be in the same local directory as this notebook file!
        with open(file_reference, 'r+', encoding="utf-8") as file:
            file_contents = file.read()

            file_contents = [ord(c) for c in file_contents] # Convert message into list
 of numbers based of ASCII values
            plain_text = []

            for i in range(len(file_contents)):
                plain = (file_contents[i]**private_key[0]) % private_key[1]
                plain_text.append(plain)

            plain_text = [chr(num) for num in plain_text] # Convert list of ASCII RSA de
crpyted values to a list of strings
            decrypted_data = "".join(plain_text)

            # Clear txt file and upload the encrypted message back to the txt file
            file.seek(0)
            file.truncate()
            file.write(decrypted_data)

        return decrypted_data # I returned the data just so that you can see it without
 opening the txt file.
```

In [8]:
```
rsa = RSA()
```

In [9]:
```
public_key, private_key = generate_lock_and_key() # see generate_lock_and_key() function
for user-input instructions !!!
print(public_key, private_key) # 'p' and 'q' MUST be greater than 10 for this to work
 !!!
```

```
Enter a number for 'p' (p > 10): 11
Enter a number for 'q' (q > 10): 13
[7, 143] [103, 143]
```

In [10]:
```
cipher = rsa.encrypt("test_file.txt", public_key)
print(cipher)
```

```
□>□□-b%-1□dcb□[vPbvPb;b□>P□bwv□>bw-1b□[>bE□Ab>!,1y□□v-!□d>,1y□□v-!b;□&-1v□[□n
```

In [11]:
```
plain = rsa.decrypt("test_file.txt", private_key)
print(plain)
```

```
Hello world, this is a test file for the RSA encryption/decryption algorithm!
```