**Assignment 2**

Name: Hongzhi Fu Student ID: 1058170

(a) To achieve higher performance of calculating exponential modulo operation, we need to create a function called fast\_expo\_mod , which takes arguments , , and returns .

def fast\_expo\_mod(a, b, n):

# fast calculation of a ^ b % n

mod = 1 # initialization

while b > 1:

if b & 1 == 1: # take the last digit of b in binary representation

mod = mod \* a % n

a = a \* a % n

b = b >> 1 # shift 1 bit to the right

return mod \* a % n

Then, we can implement function sign.

def sign(m, d, n):

return fast\_expo\_mod(m, d, n)

(b) Having implemented fast\_expo\_mod , we can easily write function verify .

def verify(s, e, n, m):

return m == fast\_expo\_mod(s, e, n)

(c) blind\_sign takes auguments , , , and (a multiplicative argument) as input, compute as , and then obtain signature . The final step is derive forged signature: .

def blind\_sign(m, e, d, n, x):

m\_prime = x \*\* e \* m # compute m'

s\_prime = sign(m\_prime, d, n) # compute s'

inverse\_x = inverse\_modulo(x, n) # compute inverse modulo of x

return inverse\_x \* s\_prime % n

(d) After calling function sign, we have the signature:

8426120691143044372772546209438038881937443238984971104217602070904916078543183081813766161223036788662505115914575633650478349043284206490480901118634760693935986808382674269152621698063491604189043947671485300746547247867490452

(e) original message signature:

463537989197757726080618016081027098741354387880292271435773264081119374791313030748577134416706490834108073135225971455228203713672286497977055891978399277918043028466260885152035443156020115664139443188024488006400963600865142780

blinded message signature:

8426120691143044372772546209438038881937443238984971104217602070904916078543183081813766161223036788662505115914575633650478349043284206490480901118634760693935986808382674269152621698063491604189043947671485300746547247867490452

1. Since the plaintext space is small (26 characters), the opponent can apply chosen plaintext attack by encrypting each character and match the incoming ciphertext. For example, if Bob sends a message encrypted as , the attacker can encrypt each 26 characters and match it with ciphertext respectively. If it matches, the attacker can know what plaintext is.
2. One countermeasure is instead of encrypting each character, we encrypt a block of characters by concatenating each character. Thus, the plaintext space is exponentially proportional to block size, and it is not practical for attackers to recover the message by chosen plaintext attack.

(a) Since and are relatively prime numbers, then we can find and such that:

The above equation can be effectively calculated by Extended Euclidean Algorithm.

Having and , we can recover original message by:

(b) One simple strategy is Jiajia requests Jaiden’s public key and since Jiajia knows , she can easily derive , where is a coefficient that can be computed by the following formula:

where // is floor division, e.g. 15 // 4 = 3.

Having the coefficient , we can derive .

For example, if , , , , , we can derive by , and finally get , which is exactly equal to .

Finally, The private key of Jaiden can be calculated by .

4.

(a) Step 1: Initiator A asks responder B to establish connection between them by transmitting identity of A() concatenated with a unique identifier, Nonce A, which encrypted by the master key of A.

Step 2: After receiving identity message of A, responder B also generates its unique number , and concatenate it with B’s identity encrypted by B’s master key, then sends a request message which contains both identity information of both A and B to KDC for a session key between them.

Step 3: KDC responds with a message containing two parts: one is encrypted by B’s master key, thus B can verify the authenticity of session key and the previous information and are not replayed by any other opponents. Similarly, another part is encrypted by A’s master key which contains , and . Note that this message cannot be tampered by others because only A knows its master key.

Step 4: After verifying the authenticity of session key from KDC, B sends , and to A. Thus, A can verify the message is indeed from KDC by comparing received with original one. Meanwhile, A also knows B’s identity.

(b) A and B can compare received nonce number and with their corresponding original one respectively, and sometimes the nonce number is timestamp which generated at a particular time thus impossible to be replayed. If two number are identical, responders can convince themselves that the key is fresh and not replayed.

(c) Because only A and B knows its corresponding master key and , which is distributed by KDC before session key request. If neither nor is compromised, A and B know that the session key distributed is not available to other opponents.

(d) No, A and B are not authenticated with each other. In step 1 and 4, two participants only send an identity to each other, which is not enough to ensure the message is indeed coming from someone. One such attack is by interception and masquerade. Suppose there is an intruder D, which intercepts the message in step 1, then generates their own nonce number , requests KDC to obtain session key and sends back to A, so A will think I have shared session key with B, but actually with D, which cannot ensure the authenticity of both A and B.

5.

(a) It satisfies on condition that unfilled block is padded with some special symbols. Since any message can be broken into several blocks, this hash function can take a variable-length message as input.

(b) It satisfies. Since the block size is fixed, and the result is constructed by XOR all encrypted blocks, the hash function always outputs a fixed-length message.

(c) Since the calculation involved only includes exponent modulus and XOR, the efficiency can be guaranteed.

(d) It satisfies, because if we want to recover the original message, then each block of hash value is constructed by RSA algorithm encrypted by public key, so it involves solving RSA problem, which is computationally infeasible to handle.

(e) It does not satisfy. Suppose the hash value of original message is , we can easily find an alternative one that all blocks of hash value is 0 except the one which is equal to , i.e. , . Thus, , and our problem is reduced to solve one block RSA problem. If the block size is small enough, it is not difficult to recover .

(f) It does not satisfy. One simple example is that one message contains only one block, which we say , and another message is three blocks of . The reason for that is the XOR result of any two or even number of identical blocks are all 0 and any message XORed with all 0 message does not change. Thus, we can conclude that it is easy for us to find two different messages that share the same hash code.