The University of Melbourne
School of Computing and Information Systems
COMP90043 Cryptography and Security

Assignment 2, Semester 2 2020
Hints and Solutions

Questions

1. [18 marks] We discussed in the subject that a message M (encoded as an integer) can be signed using RSA private key by $S = M^d \bmod n$ and verified by the corresponding public key by $M' = S^e \bmod n$ and check whether $M == M'$. We also showed the concept of blinding on RSA.

   Perform the following implementation tasks in a language of your choice. You are not allowed to use any library function to perform exponentiation. In order to get full marks, your algorithm has to be able to work in realistic cryptographic environments (consider inputs in the order of 10^{1000}).

   (a) Implement the function Sign(M, n, d) which takes the encoded message and a private key as arguments, calculates and returns the signature. You may assume that $M < n$. Print the code here.

   Solution:
   The signing is done by $S = M^d \bmod n$. A linear approach for calculating exponentiation is practically too slow for a large exponent d. If d is an even integer, we have $S = (M^{\frac{d}{2}} \bmod n)^2 \bmod n$. If d is an odd integer, we have $S = (M^{\lfloor \frac{d}{2} \rfloor} \bmod n)^2 \times M \bmod n$. Either case, we may recursively calculate $(M^{\lfloor \frac{d}{2} \rfloor} \bmod n)$ using the same idea. Thus we have an $\Theta(\log d)$ approach for exponentiation. A Python version of implementation is attached.

```
def Sign(M, n, d):
    if d == 1:
        return M % n
    temp = Sign(M * M % n, n, d // 2)
    if d % 2:
        return temp * M % n
    else:
        return temp % n
```

   (b) Implement the function Verify(S, n, e, M) takes as inputs a signature, a public key and the original message. It should return True if the signature is valid, or False if the signature is invalid. You may reuse the Sign() function implemented above, if you want to. Print the code for this function.

Solution:
The verification is done by checking whether ($S^e \bmod n$) equals to M.

```
def Verify(S, n, e, M):
    return Sign(S, n, e) == M
```

(c) Implement a function BLINDSIGN to sign a message using the blinding concept. Remind that you shouldn't directly sign the original message by $M^d \bmod n$ as you did in SIGN function, but you may call the SIGN function if needed. Print the code here.

Solution:
As discussed in the lecture, we may first create a blinded message $M' = x^e \times M \bmod n$ by selecting an integer x such that $GCD(x, n) = 1$ (this would ensure that we can find $a = e^{-1} \bmod n$). The signature of the blinded message M' can be then generated as $S' = M'^d \bmod n$. The signature of the original message can be then calculated as $a \times S' \bmod n = a \times (M'^d) \bmod n = a \times (x^e \times M)^d \bmod n = a \times x^{de} \times M^d \bmod n = ax \times M^d \bmod n = M^d \bmod n$. A Python implementation is attached (you may find an implementation of the INVERSE(x, n) function in assignment 1).

```
def BlindSign(M, n, d, e, x):
    M_prime = M * Sign(x, n, e) % n
    S_prime = Sign(M_prime, n, d)
    S = S_prime * Inverse(x, n) % n
    return S
```

An integer encoded message M and a pair of RSA keys are provided as following:
$M = 314159265?????93$ (please replace ????? with the last five digits of your student ID)
Private key: $< n, d >$ Public key: $< n, e >$
$n =$ 1139631134290681913324518075250462509444792614577115360833700594253534083115108212461164873379591734542309312064780949257819665132832661342154198437454459926525649486600336464897081397167045104842672493488133506984881500857942197501
$e = 65537$
$d =$ 2072957680681022794565143350330464253031321659272440333933281166989087050798053771266543548767583665330861850424073864444696973004489931710794150224779958495944479817291689146397299649575294462296501865902209905922547000385620583 05

(d) Sign the message by calling your above implemented SIGN function. Print the signature here.

Solution:
Take student number 999999. The signature should be 911464341470366865
3626167933447550208728104346041641010839628020307797609777930097805
2606188417231514134833566269886254543248287279363358941202064245892
9619081098162611853130709931171817431216229254305902916464432739803
012846989428.

(e) Sign the message through blinding process by calling BLINDSIGN, with x equals to the last four digits of your student ID (you may find the definition of x from week 5 lecture). Print signatures for both the blinded message and original message.

Solution:
Take student number 999999. The signature for blinded message is 10176943
9615025904634268876256765677224996833845306151440405260254767345431
9297435446003632438779634944397092726753769955341778007495073236694
9962352122876628158361302874418664185385763502764946187429865807250
4867927964493293875075.
The signature for the original message is same as the signature generated in (d).

2. [9 marks] Assume that Alice has chosen a large RSA modulus n such that factorization is impossible with reasonable time and resources. She also then chooses a large random public exponent $e < n$ for which the RSA problem is also not practical. However Bob decides to send a message to Alice by encrypting each alphabet character (represented by an integer between 0 and 25) separately using Alice's public key $< n, e >$.

(a) Describe an efficient attack against this method.

Solution:
Chosen plaintext attack can be applied here. Consider there are only 26 possible plaintexts (characters), one can effectively encrypting all of them to get corresponding ciphertexts. This would create 26 pairs of plaintext-ciphertexts. Any message (encrypted character) sent from Bob can be directly looked up from these pairs.

(b) Suggest a countermeasure to this attack.

Solution:
It's essential to increase the input size. This can be done by encrypting multiple characters at a time, or add some random padding before encryption.

3. [16 marks] Professor Parampalli generated two pairs of RSA keys for his tutors, using a pair of p and q. The chosen public component e_1 and e_2 are different prime numbers. Both p and q are very large prime numbers and were destroyed immediately after generating the following keys:

$$\text{Jaiden: } < n, e_1 >, < n, d_1 >$$
$$\text{Jiajia: } < n, e_2 >, < n, d_2 >$$

Answer the following questions with detailed process and/or justification.

(a) Lianglu wants to send a confidential message M to both Jaiden and Jiajia, so he calculates and then sends $C_1 = M^{e_1} \bmod n$ and $C_2 = M^{e_2} \bmod n$. Explain how you may recover this message without knowing Jaiden's or Jiajia's private key.

Solution:
Since $GCD(e_1, e_2) = 1$, there exist $x, y \in \mathbf{Z}$, such that $e_1 x + e_2 y = GCD(e_1, e_2) = 1$. The message can be recovered by

$$C_1^x C_2^y \bmod n = M^{e_1 x} M^{e_2 y} \bmod n = M^{e_1 x + e_2 y} \bmod n = M^1 \bmod n = M.$$

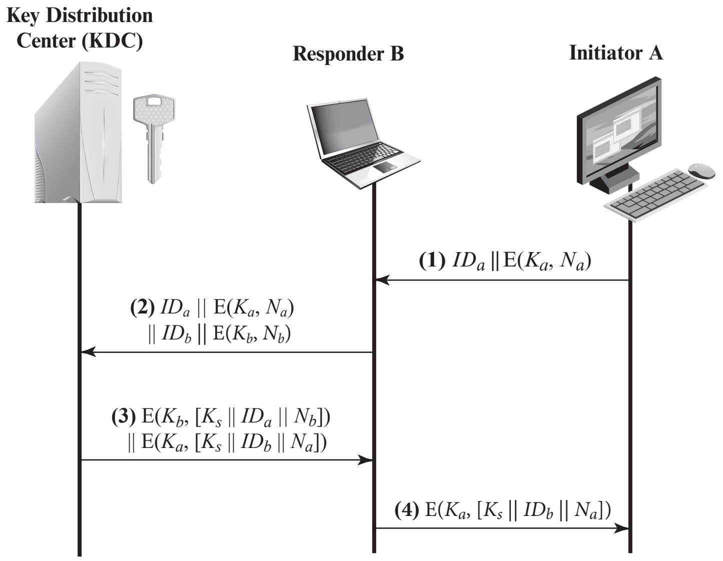
Note that one of x and y is negative. If x is negative, we need to find the multiplicative inverse of C_1 under modulo n, and then calculate $(C_1^{-1})^{-x}$.

(b) Outline a strategy that may help Jiajia recover Jaiden's private key.

Solution:
Jiajia may first calculate $\varphi'(n) = e_2 \cdot d_2 - 1 = k \cdot \varphi(n)$. She may then calculate $d_1' = e_1^{-1} \mod \varphi'(n)$.
d_1' can be directly used as Jaiden's private key, as

$$d_1' \cdot e_1 = l \cdot \varphi'(n) + 1 = l \cdot k \cdot \varphi(n) + 1.$$

4. [18 marks] An alternative key distribution method suggested by a network vendor is illustrated in the figure below.


Key Distribution Center (KDC)
Responder B
Initiator A
(1) $ID_a \| E(K_a, N_a)$
(2) $ID_a \| E(K_a, N_a) \| ID_b \| E(K_b, N_b)$
(3) $E(K_b, [K_s \| ID_a \| N_b]) \| E(K_a, [K_s \| ID_b \| N_a])$
(4) $E(K_a, [K_s \| ID_b \| N_a])$

(a) Describe the scheme in steps.

Solution:
1. A sends the message includes its identity ID_a and an encrypted nonce N_a to B, request to start the communication. The nonce is encrypted using the master key K_a.
2. B issues a request to the KDC for a session key to protect a logical connection to A. The message includes two items received from A, plus the identity of B ID_b, and a nonce N_b encrypted using B's master key K_b.
3. The respond from the KDC has two components. The first component is encrypted using K_b, and contains the session key K_s, the nonce N_b and identity of A ID_a. The second component contains the same session K_s, the nonce N_a and identity of B ID_b, and the message is encrypted by K_a.
4. B will forward the second component of the response to A.

After these four steps, A and B share the same session key.

(b) How do A and B know that the key is freshly generated?

Solution:
In step 3 and 4, both A and B receive a message encrypted by their corresponding master key. By successfully decrypting the message, they know the message is originated at the KDC. Since the correct nonce is presented in the message, the session key K_s must be generated by KDC after receiving the nonce.

(c) How could A and B know that the key is not available to other users in the system?

Solution:
Assuming that the KDC will faithfully follow the protocol, and both communication parties A and B are honest and will not share their key with others. The KDC would only issue two messages containing the session key K_s (in step 3). These two messages were encrypted using master keys of the two communication parties. Thus only A and B could decrypt the message and retrieve the session key K_s.

(d) Does this scheme ensure the authenticity of both A and B? Justify your answer.

Solution:
The authenticity is not guaranteed.
Without further communication, B could not confirm the identity of A. The only message sent from A to B was in the first step, where the message could be a replayed message from an attacker C.
Similarly, A couldn't know that B is authentic. In the second step, the attacker C could forge the message by concatenating received $ID_a||E(K_a, N_a)$ from A, and $ID_b||E(K_b, N_b)$ from a previous communication from B (to any other user in the system). KDC would not be able to identify the reply attack in this case. The attacker could then forward the second half of the message from KDC to A, which contains the valid nonce N_a.

5. [14 marks] Consider the following hash function based on RSA. The key $< n, e >$ is known to the public. A message M is represented by blocks of predefined fixed size $M_1, M_2, M_3, ..., M_m$ such that $M_i < n$. The hash is constructed by XOR the results of encrypting all blocks. For example, the hash value of a message consisting of three blocks is calculated by

$$H(M) = H(M_1, M_2, M_3) = (M_1^e \bmod n) \oplus (M_2^e \bmod n) \oplus (M_3^e \bmod n)$$

Does this hash function satisfy each of the following requirements? Justify your answers (with examples if necessary).

(a) Variable input size

Solution:
Yes. An arbitrary long message could be divided into several blocks. This hash function can generate a hash value by processing the input message block by block.

(b) Fixed output size

Solution:
Yes. The binary representation of the output has the same size of n.

(c) Efficiency (easy to calculate)

Solution:
Depending on how we define efficient. If exponentiation is considered a costly operation, this hash function could be inefficient on long messages.

(d) Preimage resistant

Solution:
Yes, as long as the RSA problem is hard to solve.
For a given hash value, finding its preimage consisting one block is same as decrypting an RSA-encrypted message without knowing the private key.
It is also hard to construct a multi-block preimage for a given hash value. The hash value is calculated by XOR-ing multiple RSA outputs. As long as the hash value itself is non-trivial (0 or 1), there is at least one RSA output being non-trivial, such that the RSA problem applies.

(e) Second preimage resistant

Solution:
No. For any message $M = \{M_1, M_2, ..., M_n\}$, we may append a block of zeros at the end of the message M to construct a new message M'. We can show that M and M' share the same hash value.

$$H(M') = (M_1^e \bmod n) \oplus (M_2^e \bmod n) \oplus ... \oplus (M_n^e \bmod n) \oplus (0^e \bmod n) = H(M)$$

(f) Collision resistant

Solution:
No. Since it does not satisfy the second preimage resistant property, it could not satisfy collision resistant requirement. A collision can be easily found by the strategy discussed in (e).