Zelong Li
24569650
Discussion 102
TA: Qi Zhong
CS161 Homework 2

1.
a.
$500^{-1} \bmod 10007$
$10007 = (500) \times 20 + 7$
$500 = (7) \times 71 + 3$
$7 = (3) \times 2 + 1$
Therefore, we have the following:
$1 = 7 - 3 \times 2$
$= 7 - 2 \times (500 - 7 \times 71)$
$= 7 \times 143 - 2 \times 500$
$= (10007 - 500 \times 20) \times 143 - 2 \times 500$
$= 143 \times 10007 - 2862 \times 500$
Finally, we have:
$500^{-1} \equiv (-2862) \bmod 10007$
Thus, $500^{-1} \equiv 7145 \bmod 10007$

b.
From Euler-Fermat theorem, we know:
$500^{(10007-1)} \equiv 1 \bmod 10007$
$500^{10006} \equiv 1 \bmod 10007$
Therefore, we have:
$500 \times 500^{10005} \equiv 1 \bmod 10007$
And 500^{10005} is the multiplicative inverse of 500 in modulo 10007
$500^{1} \equiv 500 \bmod 10007$
$500^{2} \equiv 9832 \bmod 10007$
$500^{4} \equiv 9832^{2} \bmod 10007 \equiv 604 \bmod 10007$
$500^{8} \equiv 604^{2} \bmod 10007 \equiv 4564 \bmod 10007$
$500^{16} \equiv 4564^{2} \bmod 10007 \equiv 5529 \bmod 10007$
$500^{32} \equiv 5529^{2} \bmod 10007 \equiv 8463 \bmod 10007$
$500^{64} \equiv 8463^{2} \bmod 10007 \equiv 2270 \bmod 10007$
$500^{128} \equiv 2270^{2} \bmod 10007 \equiv 9302 \bmod 10007$
$500^{256} \equiv 9302^{2} \bmod 10007 \equiv 6682 \bmod 10007$
$500^{512} \equiv 6682^{2} \bmod 10007 \equiv 7897 \bmod 10007$
$500^{1024} \equiv 7897^{2} \bmod 10007 \equiv 8992 \bmod 10007$
$500^{2048} \equiv 8992^{2} \bmod 10007 \equiv 9511 \bmod 10007$
$500^{4096} \equiv 9511^{2} \bmod 10007 \equiv 5848 \bmod 10007$
$500^{8192} \equiv 5848^{2} \bmod 10007 \equiv 5185 \bmod 10007$
$500^{10005} \equiv 500^{(1+4+16+256+512+1024+8192)} \bmod 10007 \equiv 500*604*5529*6682*7897*8992*5185 \bmod 10007 \equiv 7145 \bmod 10007$
Thus, $500^{-1} \equiv 7145 \bmod 10007$ (19 multiplication used).

2.

a.

According to this protocol, A sends $m' = m^r \bmod p$ to B and B sends $m'' = (m')^t \bmod p = m^{rt} \bmod p$ to A; then, A sends $m''' = (m'')^s \bmod p = (m^{rt})^s \bmod p = m^{rst} \bmod p$ to B. When B receives the message, the message he gets is $m^{rst} \bmod p$. Since B has the value of u, he can verify the message with u. When B verifies the message, he needs to compute $(m''')^u \bmod p = m^{rstu} \bmod p$.

We already know:

$rs = 1 \pmod{p-1}$ and $tu = 1 \pmod{p-1}$;

Thus, we let $rs = k*(p-1) + 1$ and $tu = l*(p-1) + 1$ for $k, l \in \mathbb{Z}_p$

Therefore, $m^{rs} = m^{k*(p-1)+1} \bmod p = m^{k*(p-1)} \times m \bmod p = (m^{p-1})^k \times m \bmod p$;

According to Fermat-Euler Theorem, $m^{p-1} \equiv 1 \bmod p$.

Then, we have $(m^{p-1})^k \times m \bmod p = m \bmod p$ and therefore $m^{rs} = m \bmod p$;

Similarly, $m^{tu} = m \bmod p$.

Thus, $m^{rstu} = (m^{rs})^{tu} \bmod p = m^{tu} \bmod p = m \bmod p$.

Also, if another guy other than A or B captures the encrypted message, he would get $m^r \bmod p$, $m^{rt} \bmod p$, or $m^{rts} \bmod p$. If he gets $m^r \bmod p$, without knowing r or s, he would not be able to recover m. If he gets $m^{rt} \bmod p$, without knowing rt or su, he would not be able to recover m. If he gets $m^{rts} \bmod p$, without knowing rts or u, he would not be able to recover m. Also, since compute discrete logarithm is hard, third party would not be able to compute s from these messages. Thus, the protocol's encryption and decryption work.

Because of that, B can recover the message that A sends to him and this protocol works.

b.

Suppose the man in the middle is called Mallory (M). When A wants to send B a message, she sends $m^r \bmod p$ to B. However, M intercepts and captures the message $m^r \bmod p$. M also picks a random $x \in \mathbb{Z}_p$ and sets y such that $xy = 1 \pmod{p-1}$. Then M forges a message n and sends $n^x \bmod p$ to B. Since M is man in the middle, B can't know the message he receives gets forged. Then, B tries to $n^{xt} \bmod p$ to A. But M still captures the message and sends $m^{rx} \bmod$ to A instead. After A receives the message, she then sends $m^{rxs} \bmod p = (m^{rs})^x \bmod p$ back. This time, M again gets the message and he can compute $m^{rsxy} \bmod p$ to recover A's message (according to part a). At the same time, M can send $n^{xty} \bmod p$ to B. B then recovers the message by $n^{xytu} \bmod p$ and therefore recovers the forged message n.

c.

When, A sends $m' = m^r \bmod p$ to B, the eavesdropper gets the message m'. And then, B sends $m'' = (m')^t \bmod p$ to A, the eavesdropper again gets the message m''. Then when A sends $m''' = (m'')^s \bmod p$ back to B, the eavesdropper captures the message m'''. If this eavesdropper can compute discrete logarithm, he could compute s from $m''' = (m'')^s \bmod p$ and m''. Now, after the eavesdropper gets s, he could simply compute $(m')^s \bmod p = m^{rs} \bmod p = m$ and therefore recovers the message (same reasoning as part a). Thus, if an eavesdropper can compute discrete logarithm, this protocol no longer works.

3.

a.

h'() is not pre-image resistant. When $0 \leq x \leq 2^n$, we get a h'(x). In this situation, if we ignore the left-padded zeros, we get the binary representation of x and therefore we get x. Thus, h'() is not pre-image resistant.

b.

h'() is second pre-image resistant. Given x, we can calculate h'(x). If $0 \leq x \leq 2^n$, we get a binary representation of x left-padded with zeros. In this case, in order to have h'(y) = h'(x), y and x have to have the same binary representation and therefore y = x. If $2^n \leq x$, h'(x) we get is just 1 concatenating with h(x). In this situation, h(x) is second pre-image resistant, so we still can't find $y \neq x$ such that h(x) = h(y) to make h'(y) = h'(x). In general, h'() is second pre-image resistant.

c.

h'() is collision resistant. Without loss of generality, assume $0 \leq x \leq 2^n$ and $2^n \leq y$. In this case, h'(x) outputs an n bits string starting with 0. However, h'(y) outputs an n bits string starting with 1. It's obvious that $h'(x) \neq h'(y)$. If $0 \leq x \leq 2^n$ and $0 \leq y \leq 2^n$ with $x \neq y$, since x and y are not equal, they have different binary representation. Therefore, $h'(x) \neq h'(y)$. If $2^n \leq x$ and $2^n \leq y$, since h() is collision resistant, $h(x) \neq h(y)$ and therefore $h'(x) \neq h'(y)$. Thus, in general, h'() is collision resistant.

d.

Second pre-image resistance does not imply pre-image resistance. In this question, our hash function is a good example, because it's second pre-image resistant but not pre-image resistant. In a general case, if h() is second pre-image resistant, which means given x, we can't find $y \neq x$ such that h(x) = h(y). But even if so, if the hash function's output is unique with unique input, it is possible to get the input with its output. Another example is h(x) = x.

e.

Collision resistance does not imply pre-image resistance. Again, the hash function in this question is an example that is collision resistant but not pre-image resistant. Collision resistance just makes sure that unequal inputs would not have the same output value. But even if so, if the hash function's output is unique with unique input, it is possible to get the input with its output. Another example is h(x) = x.