HW3

Yi Xue 04/10/2016

1. (a) Including IP address, will help to counter the replay attack, such as someone try to access TGS from a different computer pretending he is the valid client.

(b) Lifetime is too long, making it vulnerable for replay attack, such that BG can reuse the ticket to get the service. Lifetime is too short, client has to frequently request service ticket, causing extensive burden on TGS server.

(c) Authenticator is used to prove client's identity. It prevents BG from forging legitimate user's network address , then doing replay attack and gaining the unlimited access to the resources.

2. (a) Essentially, if k is reused, which could be identified by identical r value of two different messages, k could be recovered by the known s and H pairs. If k is recovered, x could be recovered. Then the particular digital signature is hacked.

(b) Since k is reused, r is identical.

$s_1 k = [H(M1) + x r] \bmod q \quad (1)$
$s_2 k = [H(M2) + x r] \bmod q \quad (2)$

(1) – (2)

$(s_2 - s_1) k = [H(M2) – H(M1)] \bmod q$

$k = [H(M2) – H(M1)] / (s_2 - s_1) \bmod q$

$k = (9 – 2) / (10 – 1) \bmod 11 = 7 * 9^{(-1)} \bmod 11 = 35 \bmod 11 = 2$

given H(M3) = 6

$k = [H(M3) – H(M1)] / (s_3 - s_1) \bmod q$
$2 = (6 – 2) * (s_3 - 1)^{(-1)} \bmod 11$
$(s_3 - 1)^{(-1)} \bmod 11 = 6$
$(s_3 - 1)^{(-1)} = 6$
$(s_3 - 1) * 6 \bmod 11 = 1$
$s_3 - 1 = 2$
$s_3 = 3$

3. (a) Using nonce, there is overhead to maintain the records of nonce used before. Using timestamps, communication parties should synchronize the time, also the time window allowed for the timestamp may pose some risks for replay attack.

(b) Compression after signature, you can decompress the data to retrieve the data and its signature for direct comparison, which is straightforward and does not involve other algorithm/process for the non-repudiation. Signature after compression, the compressed data and its signature requires additional decompression to verify, the indirectness is not desired.

(c) Web of trust is a decentralized system, which is based on reference between users. The advantages include the robustness (characteristics of distributed system), easy to scale up similar to Internet. The disadvantages include the info may not be updated, insecure data may get in.
X.509 certificates is centralized, and is based on the initial trust of the authority. The advantage is that the certificates can be maintained in an accurate, updated and secure fashion. The disadvantage is that if the initial CA is compromised, the whole CA chain is affected.

(d) No. After SSL handshake, all communications are encrypted, and the key is only used for this SSL communication. Next time, the key will change. Thus, BG cannot get the password as it is encrypted, BG also cannot replay it next time, as the key will change.

(e) For each SSL communication , the client will choose a premaster secret (vary each time), , combined with two random numbers picked by client and server , the server creates all required secret keys for communication. Thus, each SSL communicate will use different set of secret keys, thus preventing replay attack.

4.(a) No

(b) The collision ratio of the hash function is too high, for both weak and strong collision resistance.
e.g. If M = 101, 101 mod 91 = 10,
thus for any M', M' = 10 + 91 n (n = 0, 2, 3..) , will hash to the save value as M.

BG can replace the M with M', and sending the message M', E(h(M)) to persuade B to accept.

Programming problem:
(1) Code can perform DSA signature and verification.
Code is written in Python

DSA.py

```
#!/usr/bin/python3

import math
import time

#p, q, h, x, k, a “real” value H(M1), and a “fake” value H(M2)

def moduloPower(base, power, modulus) :
        if base == 0 :
                return 0

        answer = 1
        for i in range (0, power) :
                answer = answer * base % modulus
        return answer

def inverseMod(a, b) :


        #base case 1:  coprime
        if a % b == 1 :
                coeff = (1, a // b * (-1))
                return coeff

    #basis case 2: not coprime
        if a % b == 0 :
                print("a and b are not coprime, return 0,0")
                return (0, 0)

    #recursive cycles
        r = a % b
        q = a // b * (-1)

        coeff = inverseMod(b, r)
        newR = coeff[1]
        newQ = coeff[0] + coeff[1] * q
        return (newR, newQ)


#calculate g
def calcg(p,q,h) :
        g = moduloPower(h, (p-1)//q, p)
        print ("g = ", g)
```

```
        return g

#calculate y
def calcy(g,x,p) :
        y = moduloPower(g,x,p)
        print("y = ", y)
        return y

#calculate signature
def calcsig(g,k,p,q,x,H) :
        r = moduloPower(g, k, p) % q

        if k == 1 :
                kinverse = 1
        else :
                kinverse = inverseMod(q, k)[1]
                if kinverse < 0 :
                        kinverse += q
        s = kinverse * (H + x*r) % q
        print("(r, s) = (", r,",", s, ")" )
        return (r, s)


#perform signing process
def sign(p, q, g, x, k, H) :
        return calcsig(g,k,p,q,x,H)


#perform verifying process
def verify(p,q,g,y,r,s,H) :
        if s == 1 :
                w = 1
        else :
                w = inverseMod(q, s)[1]
                if w < 0:
                        w += q
        u1 = H * w % q
        u2 = r * w % q
        v = (moduloPower(g,u1,p) * moduloPower(y,u2,p))%p % q
        print("w = ", w)
        print("u1 = ", u1)
        print("u2 = ", u2)
        print("v = ", v)
        if v == r :
                print("Signature verfied")
                return True
        else :
                print("Signature not verfied, message faked")
                return False
```

```
print("DSA demo:")

while True:

        userInput = input("Please input p, q, h, x ,k, H(M1) and H(M2), separeted by space : ")
        p = int(userInput.split(" ") [0])
        q = int(userInput.split(" ") [1])
        h = int(userInput.split(" ") [2])
        x = int(userInput.split(" ") [3])
        k = int(userInput.split(" ") [4])
        H1 = int(userInput.split(" ") [5])
        H2 = int(userInput.split(" ") [6])

        print("Sign message H1 :")
        g = calcg(p, q, h)

        y = calcy(g, x, p)
        (r, s) = sign(p, q, g, x, k, H1)


    # s cannot be zero, otherwise, verification cause zero division failure
        if s == 0 or t == 0 :
                print("Please choose a different k to make s and t nonzero")
                continue

        print("\nNow verify signature of H1")
        verify(p,q,g,y,r,s,H1)

        print("\nNow verify signature of H2")
        verify(p,q,g,y,r,s,H2)

        quit = input("Do you want to quit ? y for quit, all other keys for continue :  ")
        if (quit == "Y" or quit == "y") :
                break
```

Output:

```
yi@yi-Inspiron-5437:~/codes/python/crypto$ ./DSA.py
DSA demo:
Please input p, q, h, x ,k, H(M1) and H(M2), separeted by space : 7 3 3 2 1 3 4
Sign message H1 :
g =  2
y =  4
(r, s) = ( 2 , 1 )

Now verify signature of H1
w =  1
u1 =  0
u2 =  2
v =  2
Signature verfied

Now verify signature of H2
w =  1
u1 =  1
u2 =  2
v =  1
Signature not verfied, message faked
Do you want to quit ? y for quit, all other keys for continue :
Please input p, q, h, x ,k, H(M1) and H(M2), separeted by space : 103 17 13 10 8 4 7
Sign message H1 :
g =  23
y =  61
(r, s) = ( 15 , 15 )

Now verify signature of H1
w =  8
u1 =  15
u2 =  1
v =  15
Signature verfied

Now verify signature of H2
w =  8
u1 =  5
u2 =  1
v =  13
Signature not verfied, message faked
Do you want to quit ? y for quit, all other keys for continue :
Please input p, q, h, x ,k, H(M1) and H(M2), separeted by space : 99981601 41659 953 82 15 22 89
Sign message H1 :
g =  77949299
y =  89355989
(r, s) = ( 20451 , 25705 )
```

Now verify signature of H1
w = 32982
u1 = 17401
u2 = 14013
v = 20451
Signature verfied

Now verify signature of H2
w = 32982
u1 = 19268
u2 = 14013
v = 30109
Signature not verfied, message faked
Do you want to quit ? y for quit, all other keys for continue : y