**Lab 10. DSA**

Ref. Appendix B, Textbook

**1** (Problem 13.2). DSA specifies that if the signature generation process results in a value of s = 0, a new value of k should be generated and the signature should be recalculated. Why?

Solution. A user who produces a signature with s = 0 is inadvertently revealing

his or her private key x via the relationship:

**2** (Problem 13.3). What happens if a k value used in creating a DSA signature is compromised?

Answer. A user's private key is discovered.

**3.** Sage Problem 1. Use Sage to solve the following problem. Assume that we are using DSA with domain parameters:

p = 7,877,914,592,603,328,881

q = 44449

g = 2,860,021,798,868,462,661

Use these domain parameters to determine if the signatures are valid in parts (a)-(c)

a) public key y = 3798043471854149631, hash value H = 59367, and signature (r,s) = (31019, 4047)

b) public key y = 1829820126190370021, hash value H = 77241, and signature (r,s) = (24646, 43556)

c) public key y = 4519088706115097514, hash value H = 48302, and signature (r,s) = (36283, 32514)

Perform a signing operation in parts (d)-(e)

d) private key x = 8146, hash value H = 22655

e) private key x = 1548, hash value H = 32782

Solution. a)

sage: p = 7877914592603328881

sage: q = 44449

sage: g = 2860021798868462661

sage: F = GF(p)

sage: y = 3798043471854149631

sage: H = 59367

sage: (r,s) = (31019, 4047)

sage: w = xgcd(s,q)[1]; w

21571

sage: u1 = H\*w % q; u1

29867

sage: u2 = r\*w % q; u2

20052

sage: v = F(g)^u1 \* F(y)^u2

sage: v = v.lift() % q

sage: v == r

b)

sage: w = xgcd(s,q)[1]; w

-15480

sage: u1 = H\*w % q; u1

31869

sage: u2 = r\*w % q; u2

30136

sage: v = F(g)^u1 \* F(y)^u2

sage: v = v.lift() % q

sage: v

24646

sage: v == r%q

True

c)

sage: y = 4519088706115097514

sage: H = 48302

sage: (r,s) = (36283, 32514)

sage: w = xgcd(s,q)[1] % q; w

3255

sage: u1 = H\*w % q; u1

6897

sage: u2 = r\*w % q; u2

172

sage: v = F(g)^u1 \* F(y)^u2

sage: v = v.lift() % q

sage: v

36283

sage: v == r

True

d)

sage: x = 8146

sage: H = 22655

sage: k = randint(2,q-1)

sage: kinv = xgcd(k,q)[1] % q

sage: r = F(g)^k

sage: r = r.lift() % q

sage: r

4382

sage: s = kinv\*(H + x\*r) % q

sage: s

19334

sage: (r,s)

(4382, 19334)

e)

sage: x = 1548

sage: H = 32782

sage: r = F(g)^k

sage: r = r.lift() % q

sage: r

36602

sage: s = kinv\*(H + x\*r) % q

sage: s

37150

sage: (r,s)

(36602, 37150)

**4.** Sage Problem 2. The purpose of this problem is to implement a DSA signature verification function.

a) Implement a function that takes domain parameters p, q, and g. Also, a Hash value H (in {1, 2, …, p-1}) a public key y, and a signature (r,s).

b) Use the function you wrote in part (a) as well as the functions from the DSA examples to simulate a DSA signature and verify like in the examples.

Solution. a)

#

# Verifies a user's DSA signature

# given p, q, g, H, r, and s

#

def DSA\_verify(p, q, g, y, H, r, s):

w = xgcd(s, q)[1] % q

u1 = H\*w % q

u2 = r\*w % q

v = F(g)^u1 \* F(y)^u2

v = v.lift() % q

return (v == r)

b)

sage: load DSAExamples.sage

sage: load DSASageExSolution.sage

sage: (p, q, g) = DSA\_generate\_domain\_parameters()

sage: (p, q, g)

(7328722685184837689, 51361, 211492003612531049)

sage: (x, y) = DSA\_generate\_keypair(p, q, g)

sage: (x, y)

(43970, 2702057011488954857)

sage: # select a random hash value to sign

sage: H = randint(2,p-1)

sage: H

4135875385774901780

sage: # sign it

sage: (r, s) = DSA\_sign(p, q, g, x, H)

sage: (r, s)

(48692, 7084)

sage: # verify that signature

sage: DSA\_verify(p, q, g, y, H, r, s)

True

Sage: # mess with the signature and the signature verification will fail.

sage: r = r - randint(1,100)

sage: DSA\_verify(p, q, g, y, H, r, s)

False

**5.** DSA Example. Using Sage, we can perform a DSA sign and verify:

sage: # first we generate the domain parameters

sage: # generate a 16 bit prime q

sage: q = 1;

sage: while (q < 2^15): q = random\_prime(2^16)

....:

sage: q

42697

sage: # generate a 64 bit p, such that q divides (p-1)

sage: p = 1

sage: while (not is\_prime(p)):

....: p = (2^48 + randint(1,2^46)\*2)\*q + 1

....:

sage: p

12797003281321319017

sage: # generate h and g

sage: h = randint(2,p-2)

sage: h

5751574539220326847

sage: F = GF(p)

sage: g = F(h)^((p-1)/q)

sage: g

9670562682258945855

sage: # generate a user public / private key

sage: # private key

sage: x = randint(2,q-1)

sage: x

20499

sage: # public key

sage: y = F(g)^x

sage: y

7955052828197610751

sage: # sign and verify a random value

sage: H = randint(2,p-1)

sage: # signing

sage: # random blinding value

sage: k = randint(2,q-1)

sage: r = F(g)^k % q

sage: r = F(g)^k

sage: r = r.lift() % q

sage: r

6805

sage: kinv = xgcd(k,q)[1] % q

sage: s = kinv\*(H + x\*r) % q

sage: s

26026

sage: # Verifying

sage: w = xgcd(s,q)[1]; w

12250

sage: u1 = H\*w % q; u1

6694

sage: u2 = r\*w % q; u2

16706

sage: v = F(g)^u1 \* F(y)^u2

sage: v = v.lift() % q

sage: v

6805

sage: v == r

True

sage: # sign and verify another random value

sage: H = randint(2,p-1)

sage: k = randint(2,q-1)

sage: r = F(g)^k

sage: r = r.lift() % q

sage: r

3284

sage: kinv = xgcd(k,q)[1] % q

sage: s = kinv\*(H + x\*r) % q

sage: s

2330

sage: # Verifying

sage: w = xgcd(s,q)[1]; w

4343

sage: u1 = H\*w % q; u1

32191

sage: u2 = r\*w % q; u2

1614

sage: v = F(g)^u1 \* F(y)^u2

sage: v = v.lift() % q

sage: v

3284

sage: v == r

True

**6.** Sage functions example. The following functions implement DSA domain parameter generation, key generation, and DSA Signing.

#

# Generates a 16 bit q and 64 bit p, both prime

# such that q divides p-1

#

def DSA\_generate\_domain\_parameters():

g = 1

while (1 == g):

# first find a q

q = 1

while (q < 2^15): q = random\_prime(2^16)

# next find a p

p = 1

while (not is\_prime(p)):

p = (2^47 + randint(1,2^45)\*2)\*q + 1

F = GF(p)

h = randint(2,p-1)

g = (F(h)^((p-1)/q)).lift()

return (p, q, g)

#

# Generates a users private and public key

# given domain parameters p, q, and g

#

def DSA\_generate\_keypair(p, q, g):

x = randint(2,q-1)

F = GF(p)

y = F(g)^x

y = y.lift()

return (x,y)

#

# given domain parameters p, q and g

# as well as a secret key x

# and a hash value H

# this performs the DSA signing algorithm

#

def DSA\_sign(p, q, g, x, H):

k = randint(2,q-1)

F = GF(p)

r = F(g)^k

r = r.lift() % q

kinv = xgcd(k,q)[1] % q

s = kinv\*(H + x\*r) % q

return (r, s)