Course:
CE 340 Cryptography & Network Security

Course Instructor:
Assoc. Prof. Dr. Süleyman KONDAKÇI

Project Title:
Final Project
DSA Implementation in sage

Submitted By:
Serdar HALİLOĞLU
20120602019

Description:

- Generate a digital signature(DS) based on the DSA algorithm.
- Sign a document using the generated DS.
- Verify that the signing is correctly done.

Outcomes:

- Learning how to use DSA algorithm in sage
- Learning how to use some functions such as converting integers into asci codes.
- Learning how to sign a document.
- Learning how to verify whether signing is correct or not.

I will explain all the lines, respectively, that I wrote.

1. I used hashlib. I will explain why we need to import this.
2. Generate Components.
    a. Choose a prime number p, which is called the prime divisor.
    b. Choose another primer number q namely prime modulus.
    c. Choose an integer g, such that $1 < g < p$, $g = h^{(q-1)/p} \bmod q$.
    p is also called g's multiplicative order modulo q.

```
import hashlib
## Choose a prime number p, which is called the prime divisor.
p = 1;
while (p < 2^15): p = random_prime(2^16)

##Choose another primer number q. q is called the prime modulus.
q=1;
while (not is_prime(q)): q = ( randint(1,2^46)*2*p)+1
q;p
```

```
6954972104331890987
54001
```

```
h = randint(2,q-2)
F=GF(q);F
g=F(h)^((q-1)/p)
```

```
6061361153652890213
4327255874027041262
```

3. Generate Private Components.
    a. Choose an integer x, such that $0 < x < p$.
        x = randint(0,p)
    b. Compute y as $g^x \bmod q$.
        y = (g**x)%q

```
## Choose an integer, such that 0 < x < p.
x = randint(0,p)

## Compute y as g**x mod q.
y = (g**x)%q
```

```
50950
5152298205921574544
```

4. Print Public and Private Key
    a. Package the public key as {p,q,g,y}.
        i. pubkey = (p,q,g,y)
    b. Package the private key as {p,q,g,x}.
        i. prikey = (p,q,g,x)
    c. pubkey;prikey

```
## Package the public key as {p,q,g,y}.
## Package the private key as {p,q,g,x}.

pubkey = (p,q,g,y)
prikey = (p,q,g,x)

pubkey;prikey
```

```
(54001, 6954972104331890987, 4327255874027041262, 5152298205921574544)
(54001, 6954972104331890987, 4327255874027041262, 50950)
```

5. Define plainText and open text file such that it is readable.
    a. plainText=open('/home/serdar/Desktop/CE340-FinalProject_serdarhaliloglu/plainText.txt','r')
    b. Define PT variable for reading .
        i. PT = plainText.read()
    c. hashlib is imported in order to generate the message digest h, using a hash algorithm like SHA1.
        i. h=hashlib.sha1(PT)
        ii. a = h.hexdigest()

```
plainText=open('/home/serdar/Desktop/CE340-FinalProject_serdarhaliloglu/plainText.txt','r')
PT = plainText.read()
h=hashlib.sha1(PT)
a = h.hexdigest()
h;a
```

```
<sha1 HASH object @ 0x7fa6e7b628a0>
'5c9a7a43506058ae8c9e7add3df9b8652f1c494b'
```

6. Ascii Function and string-integer converting
    a. Define a function called convert_to_ascii because we need to convert hex value to ascii value.
    b. Define a variable M, and assign the result of function convert to get the ascii value.
    c. Then convert string to int value.
    d. Print M.

```
def convert_to_ascii(hash):
    asci='1'
    for x in range(0,len(PT)):
        if ord(hash[x])<100:
            asci += '0' + str(ord(hash[x]))
        else:
            asci += str(ord(hash[x]))
    return asci

M= convert_to_ascii(a)
M = int(M)
M
```

```
1053099057097055097052051053048054048053L
```

7. Generate k, r, sign(=s)
    a. Generate a random number k, such that 0 < k < p.
        i. k = randint(0,p)
    b. Compute r as (g^k mod q) mod p.
        i. r = mod((g**k)%q,p)
    c. Compute s
        i. s=mod(((M+x*r)/k),p)
    d. Print k,r,s
        i. k;r;s

```
k = randint(0,p)
r = mod((g**k)%q,p)
s=mod(((M+x*r)/k),p)
k;r;s
```

```
17775
29243
14185
```

8. Define signText and open text file such that it is writeable.
    a. signText=open('/home/serdar/Desktop/CE340-FinalProject_serdarhaliloglu/signText.txt','w')
    b. Write r,s and PT into signText file and close file.
        i. signText.write(str(r))
        ii. signText.write(str(s))
        iii. signText.write(str(PT))
        iv. signText.close()

```
signText=open('/home/serdar/Desktop/CE340-FinalProject_serdarhaliloglu/signText.txt','w')
signText.write(str(r))
signText.write(str(s))
signText.write(str(PT))
signText.close()
signText
```

```
<closed file
'/home/serdar/Desktop/CE340-FinalProject_serdarhaliloglu/signText.txt',
mode 'w' at 0x7fa6e80dff60>
```

9. Package the digital signature as {r,s} and print.
    a. digital_signature = (r,s)
    b. digital_signature

```
digital_signature = (r,s)
digital_signature
```

```
(29243, 14185)
```

10. To verify a message signature, the receiver of the message and the digital signature follows steps:
    a. Define Verify and open signText file such that it is readable.
        Verify=open('/home/serdar/Desktop/CE340-FinalProject_serdarhaliloglu/signText.txt','r')
    b. Verify.read()
    c. Generate the message digest h, using the same hash algorithm.
        i. h = hashlib.sha1(PT)
        ii. b = h.hexdigest()
        iii. b
    d. Define a variable N, and assign the result of function to get its ascii value.
    e. Then convert string to int value.
    f. Print N.
        i. N = convert_to_ascii(b)
        ii. N = int(N)
        iii. N

```
## Verify
Verify=open('/home/serdar/Desktop/CE340-FinalProject_serdarhaliloglu/signText.txt','r')
Verify.read()
h = hashlib.sha1(PT)
b = h.hexdigest()
b

N = convert_to_ascii(b)
N = int(N)
N
```

```
1053099057097055097052051053048054048053L
```

    g. Compute w, such that s*w mod q = 1. w is called the modular multiplicative inverse of s modulo p.
        i. w=(s^(-1))%p;
        ii. w

```
## w is called the modular multiplicative inverse of s modulo p.
w=(s^(-1))%p
w
```

```
4298
```

h. Compute u1 = N*w mod p.
u1 = mod((N*w),p)

i. Compute u2 = r*w mod p
u2 = mod((r*w),p)

j. Print u1,u2
u2 = mod((r*w),p)

```
u1 = mod((N*w),p)
u2 = mod((r*w),p)
u1;u2
```

```
11738
26087
```

k. Compute v = ((g^{u1}*y^{u2}) mod q) mod p.
   i. v=mod((((g**u1)*(y**u2))%q),p)
   ii. v

```
v=mod((((g**u1)*(y**u2))%q),p)
v
```

```
29243
```

l. If v == r, the digital signature is valid.
v == r

```
v==r
```

```
True
```

User Guide

All steps are described below, respectively. Please follow the steps to run.

- ✓ Extract CE340-FinalProject_serdarhaliloglu.zip.
- ✓ Open sage math with sudo.
- ✓ Use notebook() for the browser-based notebook interface.
- ✓ Open http://localhost:8080 in your web browser.
- ✓ Select Upload on Sage Notebook.
- ✓ Click Browse Button, then select FinalProject.sws and upload worksheet.
- ✓ Change the location of filenames in worksheet.
- ✓ After the all steps, click evaluate all in action part.