
1: The First Problem

(a) One-time password SecureID system:

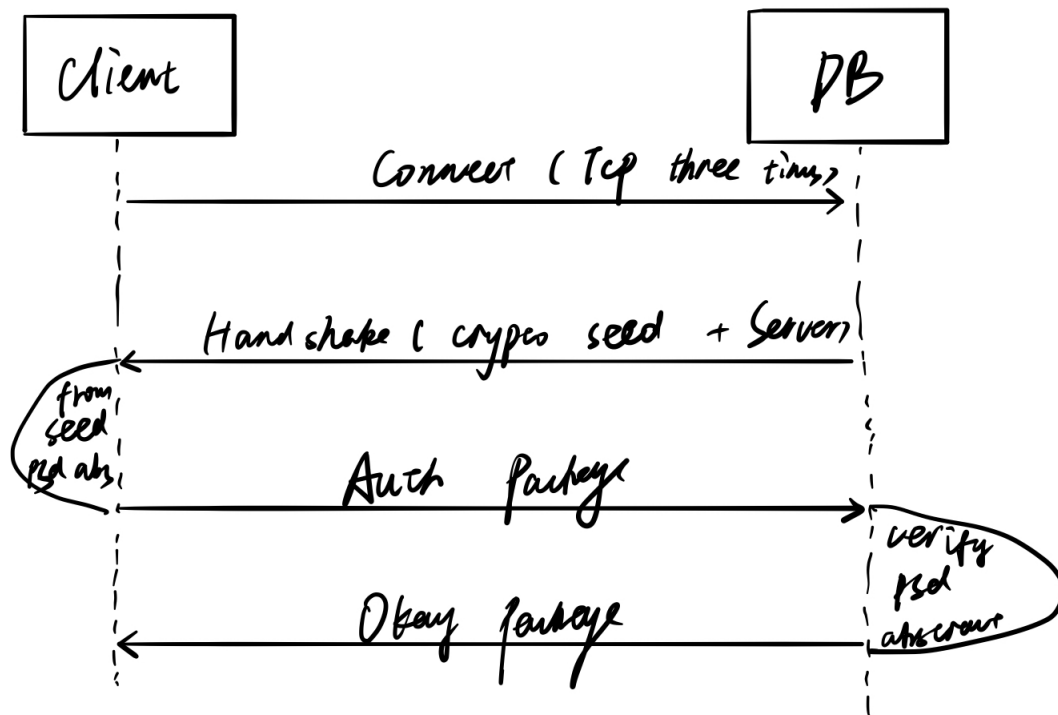
SMS is an ideal way to reach the recipient. This ensures confirmation of identity via registered mobile number. The banking industry is a perfect application of OTP SMS provider. All the transactions are money related. Such operations must process using verified details. The generation of OTP is done through using the gateway using the SMS after they are generated at the server side. The security of OTP is reinforced because both the user platform and mobile number are not vulnerable to attacks at the same time.

(b) S/Key system:

Based on the S/Key system authentication method, the user submits registration data including the user name, password, iteration value, and biometric value to the authentication server through the client when the user first registers. The first one-time password for the password sequence. In the user identity authentication process, two-way authentication is performed between the client and the server, and it is a two-way authentication combining the one-time password and biometric value of the registered user.

(b) Challenge response protocol:

Mysql handshake process, after the client actively initiates the link successfully, the server actively sends a challenge protocol, which contains the seed and the attributes supported by the server. Then the client sends the response protocol to the server for verification through the seed summary password.



2: The Second Problem

(a) Algorithm: Origin

```

import time
from fastecdsa import keys
from fastecdsa import curve
from Crypto.Hash import SHA256
cur = curve.P256

def key_gen():
    sk = keys.gen_private_key(cur)
    pk = keys.get_public_key(sk, cur)
    g = cur.G
    return sk, pk, g

def sig(m, sk):
    tmp_k = sk
    k1 = keys.gen_private_key(cur)
    k2 = keys.get_public_key(k1, cur)
    h = SHA256.new()
    x = '{:0{x}}'.format(k2.x, 64)
    y = '{:0{x}}'.format(k2.y, 64)
    h.update(m + bytes(x+y, "ascii"))
  
```

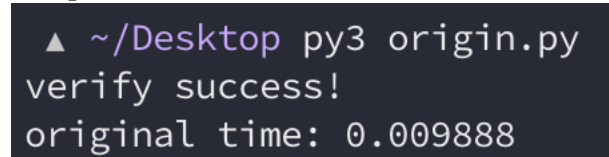
```

    ct = h.hexdigest()
    ot = k1 + tmp_k * int(ct, 16)
    return {'k2':k2, 'ot':ot}

def verify(sign, m, g, pk):
    k2 = sign['k2']
    ot = sign['ot']
    h = SHA256.new()
    x = '{:0{x}}'.format(k2.x, 64)
    y = '{:0{x}}'.format(k2.y, 64)
    h.update(m + bytes(x + y, "ascii"))
    ct = h.hexdigest()
    r1 = g * ot
    r2 = k2 + pk * int(ct, 16)
    if r1 == r2:
        print('verify success!')

begin = time.time()
sk, pk, g = key_gen()
m = b"CSCI468/968 Advanced Network Security, Spring 2020"
sign = {}
sign = sig(m, sk)
verify(sign, m, g, pk)
end = time.time()
print("original time: %f" % (end - begin))

```

Output:


```

▲ ~/Desktop py3 origin.py
verify success!
original time: 0.009888

```

(b) Algorithm: Optimize

```

import time
from fastecdsa import keys
from fastecdsa import curve
from Crypto.Hash import SHA256
cur = curve.P256

def key_gen():
    sk = keys.gen_private_key(cur)
    pk = keys.get_public_key(sk, cur)
    g = cur.G
    return sk, pk, g

def sig(m, sk):

```

```

tmp_k = sk
k1 = keys.gen_private_key(cur)
k2 = keys.get_public_key(k1, cur)
h = SHA256.new()
x = '{:0{x}}'.format(k2.x, 64)
y = '{:0{x}}'.format(k2.y, 64)
h.update(m + bytes(x+y, "ascii"))
ct = h.hexdigest()
ot = k1 + tmp_k * int(ct, 16)
return {'ct':ct, 'ot':ot}

def verify(sign, m, g, pk):
    ct = sign['ct']
    ot = sign['ot']
    h = SHA256.new()
    k2 = g * ot - pk * int(ct, 16)
    x = '{:0{x}}'.format(k2.x, 64)
    y = '{:0{x}}'.format(k2.y, 64)
    h.update(m + bytes(x + y, "ascii"))
    ct2 = h.hexdigest()
    if ct == ct2:
        print('verify success!')
    else:
        print('verify fail!')

begin = time.time()
sk,pk,g = key_gen()
m = b"CSCI468/968 Advanced Network Security, Spring 2020"
sign = {}
sign = sig(m, sk)
verify(sign, m, g, pk)
end = time.time()
print("optimize time: %f" % (end - begin))

```

Output:

```

▲ ~/Desktop py3 optimize.py
verify success!
optimize time: 0.009712

```

3: The Third Problem

The derivation process:

$$a_{z0} = a_{t0} + sk \cdot C_0 \quad (1)$$

$$a_{z1} = a \cdot a_{t0} + b + sk \quad (2)$$

$$c_0 = H(m_0, v_{t0}) \quad (3)$$

$$v_{t0} = g^{a_{t0}} \quad (4)$$

$$c_1 = H(m_1, V_{t1}) \quad (5)$$

$$v_{t1} = g^{a \cdot a_{t0} + b} \quad (6)$$

$$a \cdot a_{z0} + b = a \cdot a_{t0} + b \quad (7)$$

$$a_{z1} = a \cdot a_{t0} + b + sk \cdot c_1 \quad (8)$$

$$sk = \frac{a \cdot a_{z0} + b - a_{z1}}{a \cdot c_0 - c_1} \quad (9)$$

4: The Fourth Problem

The derivation process:

$$\bar{a} = \sum_{i=1}^n a_{zi} \quad (10)$$

$$\bar{c} = \sum_{i=1}^n c_i \quad (11)$$

$$\Rightarrow \frac{\bar{a}^1}{\bar{c}^1} \quad (12)$$

$$\bar{a} = a_{t0} + sk \cdot \bar{c} \quad (13)$$

$$\bar{a}^1 = a_{t0} + sk \cdot \bar{c}^1 \quad (14)$$

$$\Rightarrow sk = \frac{\bar{a} - \bar{a}^1}{\bar{c} - \bar{c}^1} \quad (15)$$