ECurve

Ilya Bardinov

20.12.2016

```
import hashlib

# elliptic curve domain parameters, prime192v1
#
# p = 2**192 - 2**64 - 1
# a = -3
# b = 0x64210519E59C80E70FA7E9AB72243049FEB8DEECC146B9B1
# x = 0x188DA80EB03090F67CBF20EB43A18800F4FF0AFD82FF1012
# y = 0x07192B95FFC8DA78631011ED6B24CDD573F977A11E794811
# n = 0xFFFFFFFFFFFFFFFFFFFFFFFF99DEF836146BC9B1B4D22831
# h = 1

F = FiniteField(2**192 - 2**64 - 1)
a  = -3
b  = 0x64210519E59C80E70FA7E9AB72243049FEB8DEECC146B9B1
E  = EllipticCurve(F, [a, b])
P  = E((0x188DA80EB03090F67CBF20EB43A18800F4FF0AFD82FF1012,
        0x07192B95FFC8DA78631011ED6B24CDD573F977A11E794811))
n  = 0xFFFFFFFFFFFFFFFFFFFFFFFF99DEF836146BC9B1B4D22831
h  = 1
Fn = FiniteField(n)

def digest(msg):
        msg = str(msg)
        return Integer('0x' + hashlib.sha1(msg).hexdigest())

# Algorithm Elliptic curve key pair generation
# Require:
#       generator point P of elliptic curve E
#       order n of P and the field Zn defined by n
# Input:
#       N/A
# Output:
#       keypair (Q, d)
#               public key point Q on curve E
```

```
#                private key d in [1, n-1]
#

def ec_keygen():
        d = randint(1, n - 1)
        Q = d * P
        return (Q, d)

# Algorithm signature generation
# Require:
#        generator point P of elliptic curve E
#        order n of P and the field Zn defined by n
# Input:
#        message m
#        private key d in [1, n - 1]
# Output:
#        signature (r, s) where r, s in Zn
#
# 1. Generate random k in 0<k<n;
# 2. Calculate point Q of EC, Q=k*P;
# 3. Consider r = x_Q mod n, where x_Q — x-coord of point Q. If r\
   =0, go to step 1;
# 4. Calculate digest: e=digest(m);
# 5. Calculate s = (rd+ke) mod n. if s=0, go to step 1;
#


def ecdsa_sign(d, m):
        r = 0
        s = 0
        while s == 0:
                k = 1
                while r == 0:
                        k = randint(1, n - 1)
                        Q = k * P
                        (x1, y1) = Q.xy()
                        r = Fn(x1)
                kk = Fn(k)
                e = digest(m)
                s = kk ^ (-1) * (e + d * r)
        return [r, s]

# Algorithm signature verification
# Require:
#        generator point P of curve E
#        order n of P and the field Zn defined by n
# Input:
```

```
#        public key point Q on curve E
#        message m
#        signature sig = (r, s) where r, s in Zn
# Output:
#        True or False
#
# 1. Calculate digest: e=digest(m);
# 2. Calculate w = s-1 mod n;
# 3. Calculate u1 = e*w mod n and u2 = r*w mod n;
# 4. Calculate point X = u1*P + u2*Q;
# 5. Consider R = x_X mod n, where x_X — x-coord of X;
# 6. If v=r, then its verified.
#

def ecdsa_verify(Q, m, r, s):
        e = digest(m)
        w = s ^ (-1)
        u1 = (e * w)
        u2 = (r * w)
        P1 = Integer(u1) * P
        P2 = Integer(u2) * Q
        X = P1 + P2
        (x, y) = X.xy()
        v = Fn(x)
        return v == r

# TEST

(Q, d) = ec_keygen()
m = 'signed message'
not_m = 'signed_message'

[r, s] = ecdsa_sign(d, m)
result = ecdsa_verify(Q, not_m, r, s)

print "EC Public Key         : ", Q.xy()
print "EC Private Key        : ", d
print "Signed Message        : ", m
print "Signature      : "
print " r = ", r
print " s = ", s
print "Verified Message      : ", not_m
print "Verification Result : ", result
```

EC Public Key : (57249493997082740752517310260420954920491227477575766565651328,
399198545597112798250933963651417831565891127309411183766)
EC Private Key : 6230014730089507748874836791390996675035086909716264736809
Signed Message : signed message

Signature :
r = 588526787054254298122186458257065321354932661339089782716
s = 2385859870594706328612926328003342319564042582737967231805
Verified Message : signed_message
Verification Result : False