+CryptoGraphy

Digital Signature Algorithm (DSA)

Tạo Khóa :

- Chọn số nguyên tố 160 bit q
- Chọn số nguyên tố L bit p sao cho p = qz + 1 , 512 <= L <= 1024 , L chia hết cho 64
- Chọn h, 1 < h < p - 1 sao cho g = h^z mod p > 1 (z = (p -1) / q)
- Chọn x ngẫu nhiên thỏa mãn 0 < x < q
- Y = g^x mod p
- Khóa công khai (p,q,g,y) . Khóa riêng là x .

Ký :

- Tạo một số ngẫu nhiên với mỗi thông điệp , k thỏa mãn 0 < k < q
- Tính r = (g^k mod p) mod q
- Tính s = (k^{-1}(SHA1(m) + xr)) mod q , m là thông điệp
- Tính toàn lại chữ kí khi r = 0 hoặc s = 0
- Chữ kí là (r,s)

Xác nhận :

- Loại bỏ chữ kí nếu 0 < r < q và 0 < s < q không thỏa mãn .
- W = s^{-1} mod q
- U1 = SHA1(m) * w mod q
- U2 = rw mod q
- V = g^{u1} . y^{u2} mod p mod q
- Chữ kí có hiệu lực nếu v = r .

Từ Cryptopals thì chúng ta có thể có những cách tấn công vào DSA như sau :

- **Khi biết k thì chúng ta có thể tìm được private key x theo công thức :**

$$x = \frac{(s * k) - H(msg)}{r} \mod q$$

- **Khi k được dùng lại nhiều lần thì k có thể được tính bởi công thức :**

$$k = \frac{(m1 - m2)}{(s1 - s2)} \bmod q$$

- **Thay đổi g bằng những giá trị đặc biệt . VD : g = p + 1**

VolgaCTF 2018

```
class LCG():
        def __init__(self):
                self.a = 34377762929967774679766575475779676575 47
                self.b = 82866986546959242626236347547757 4643634
                self.m = 11188172152664730994014892998359450277136352 48219
                self.seed = seed
                self.state = (self.a * self.seed + self.b) % self.m

        def next_number(self):
                self.state = (self.a * self.state + self.b) % self.m
                return self.state


generator = LCG()
signature = DSA()

for _ in range(2):
        message = "VolgaCTF{" + os.urandom(16).encode('hex') + "}"
        k = generator.next_number()
        (r, s) = signature.sign(message, k)
        print (message, r, s)
        print signature.verify(message, r, s)
```

Bài toán sử dụng thuật toán kí số và yêu cầu là tìm được private key x .

Chúng ta có thể thấy nó kí từng message và cho ta r,s và message tương ứng.

Một điểm đặc biệt là k không hoàn toàn random mà được lấy từ thuật toán LCG() .

Như vậy ta có quan hệ giữa các k :

$$K_2 = K_1 * a + b$$

Đồng thời ta sử dụng công thức của s :

K * s = MD5(m) + xr (mod q)

Như vậy với 2 message , ta có 1 hệ 3 phương trình 3 ẩn , từ đó tìm x là xong :v

Midnight Sun CTF 2019 EZDSA

```
class PrivateSigningKey:

    def __init__(self):
        self.gen = 0x44120dc98545c6d3d81bfc7898983e7b7f6ac8e08d3943af0be7f5d52264abb3775a905e003151ed0631376165b65c8ef72d0b6880da7e4b5e7b833377bb50fde65846426a5bfdc182673b6b2504ebfe0d6bca36338b3a3be334689c1afb17869baeb2b0380351b61555df31f0cda3445bba4023be72a494588d640a9da7bd16L
        self.q = 0x926c99d24bd4d5b47adb75bd9933de8be5932f4bL
        self.p = 0x80000000000001cda6f403d8a752a4e7976173ebfcd2acf69a29f4bada1ca3178b56131c2c1f00cf7875a2e7c497b10fea66b26436e40b7b73952081319e26603810a558f871d6d256fddbec5933b77fa7d1d0d75267dcae1f24ea7cc57b3a30f8ea09310772440f016c13e08b56b1196a687d6a5e5de864068f3fd936a361c5L
        self.key = int(FLAG.encode("hex"), 16)

    def sign(self, m):

        def bytes_to_long(b):
            return long(b.encode("hex"), 16)

        h = bytes_to_long(sha1(m).digest())
        u = bytes_to_long(Random.new().read(20))
        assert(bytes_to_long(m) % (self.q - 1) != 0)

        k = pow(self.gen, u * bytes_to_long(m), self.q)
        r = pow(self.gen, k, self.p) % self.q
        s = pow(k, self.q - 2, self.q) * (h + self.key * r) % self.q
        assert(s != 0)

        return r, s
```

Ta thấy k không được lấy random vậy nên chắc là lỗi ở đây rồi .

Chương trình cho phép chúng ta được kí bất kì đoạn message nào .

Đồng thời nó cũng loại bỏ đi những giá trị mà **m % (q – 1) == 0** .Đó là những giá trị làm cho k bằng 1 với mọi u .

Ta có thể có những giá trị khác để điều này luôn đúng. Chọn m = (q-1)//2 thì ta cũng có điều tương tự.

Vậy nên khi đó ta biết được k = 1 và dễ dàng recovery lại key theo các phương pháp đã học .

CSAW 2018 Final DSA

Bài này chúng ta có thể lấy được từ server **public_key** từ trên trang web của nó .

Cũng đồng thời học lỏm được cách để tương tác với server dùng **request** trong python .

```
def public_key():
        return json.loads(rq.get(url+"public_key").text)
```

Đồng thời chúng ta cũng có thể **sign** :

```
 def sign():
         return json.loads(rq.get(url+"sign/AAAA").text)
```

Hàm **sign** :

```
@app.route("/sign/<data>")
def signer(data):
    r, s = sign(ctf_key, data)
    return json.dumps({"r": r, "s": s})

def sign(ctf_key, data):
    data = data.encode("ascii")
    pn = ctf_key.private_numbers()
    g = pn.public_numbers.parameter_numbers.g
    q = pn.public_numbers.parameter_numbers.q
    p = pn.public_numbers.parameter_numbers.p
    x = pn.x
    k = random.randrange(2, q)
    kinv = _modinv(k, q)
    r = pow(g, k, p) % q
    h = hashlib.sha1(data).digest()
    h = int.from_bytes(h, "big")
    s = kinv * (h + r * x) % q
    return (r, s)
```

Nhìn xem thì mọi thứ hoàn toàn bình thường . k cũng được lấy random bằng một hàm trong python luôn @@ Vậy lỗi ở đây là gì ?

Lại thêm một cái unbleiveable là hàm random built-in trong python bị crack . Nó dùng Mersen twister để sinh số ngẫu nhiên . Nó không hề an toàn mà chỉ nhanh nên nó không được sử dụng trong cryptography .

Có một thư viện trong python là **randcrack** được thiết kế để predict random number của cách hàm : randrange, randint, getrandbits,randbelow,randchoice .

https://github.com/tna0y/Python-random-module-cracker

Có một điều kiện là hàm predict và hàm lấy random phải được thực hiện liền kề nhau , hầu như là cùng lúc thì tỉ lệ success lên tới 99% .

Nhờ vào công cụ này mà chúng ta có thể request lên server đồng thời tiên đoán luôn k , từ đó recovery lại được privatekey .

```
def getx(pk):
        h = hashlib.sha1("AAAA".encode("ascii")).digest()
        h = int.from_bytes(h, "big")
        g=pk['g']
        q=pk['q']
        p=pk['p']
        sig = sign()
        s=sig['s']
        r=sig['r']
        k=rc.predict_randrange(2,q)
        x=int((s*k-h)*gmpy2.invert(r,q) % q)
        return x
```

Hihi do méo có server để thực hành cho nên chúng ta đọc hiểu là ok :v Thực ra crypto khác với pwn là mình cần nắm được cách attack thôi còn code thì ezz cơ còn pwn thì dù nắm được cách tấn công rồi nhưng viết code vẫn méo hiểu sai ở đâu . Lắm chỗ linh tinh vcl :v

Ta có thể thây hàm k = rc.predict_randrange(2,q) được thực hiện ngay sau khi kí . Phải đảm bảo là mạng đủ nhanh , không bị gián đoạn :v

Note : Không sử dụng hàm random builtin python trong mật mã mà nên dùng các hàm trong thư viện khác như os hay crypto .

Quanh quanh thì DSA cũng chỉ có vài dạng như trên nên mình cũng không trình bày nhiều thêm nữa :v