CS 406: Cryptography and Network Security
Hands On - 4

Chaitanya Garg, 210050039

April 07, 2024

Challenge - 1

The solution is pretty straight forward with the openssl command as follows:

```
openssl pkeyutl -in cipher.bin -inkey priv.pem -decrypt -pkeyopt rsa_padding_mode:
    oaep
```

The flag comes to be : cs406{r54_043p_3t_0p3n_55l}

Challenge - 2

We can view the certificate details by using a browser and get all the details regarding the certificate, including the issuer.

We can also use the openssl command to connect to the website and get the information. The command used is as follows:

```
echo | openssl s_client -servername cse.iitb.ac.in -connect cse.iitb.ac.in:443 2>/dev
    /null | openssl x509 -noout -issuer 2>/dev/null | awk -F '=' '/CN/{print $NF}' |
    sed -e 's/ /\_/g' -e 's/.*/cs406{\0}/g'
```

The echo is just to send an empty input to the connection so that it closes immediately and does not wait for any input. The openssl s_client command creates a client and connects it to the given website. This also prints many information regarding the website, the server and their certificates. The openssl x509 command with the given flags filters out the issuer of the certificate information. This command can also be replaced by a simple `grep "issuer"` command. The awk command filter the Canonical Name of the issuer. The sed command is to replace all spaces by an underscore and enclose it in `cs406{}`.

The flag comes out to be: cs406{RapidSSL_TLS_RSA_CA_G1}

Challenge - 3

Here we use use the following formula used in ecdsa algorithm:

$$s = nonce^{-1} * (h + r * priv_k)\%order$$

Since r, $priv_k$ and $nonce$ are common across the 2 messages, by subtracting the equation for the 2 messages, we get:

$$\implies s_1 - s_2 = nonce^{-1} * (h_1 - h_2)\%order$$

$$\implies nonce = (h_1 - h_2) * (s_1 - s_2)^{-1}\%order$$

Here note that r_1 and r_2 are same, (the x coordinate of the point on the elliptical curve).

Now that nonce is recovered, we can rearrange the equation for message 1 to get the private_key as:

$$s_1 = nonce^{-1} * (h_1 + r_1 * priv_k)\%order$$

$$\implies priv_k = (s_1 * nonce - h_1) * r_1^{-1}\%order$$

The code written is as follows:

```
G = ecdsa.SECP256k1.generator
order = G.order()

h_1 = int(hashlib.sha256(msg1.encode()).hexdigest(), base=16)
h_2 = int(hashlib.sha256(msg2.encode()).hexdigest(), base=16)

nonce_rec = ((h_1 - h_2) * inverse(s_1 - s_2, order)) % order

privkey_rec = ((s_1 * nonce_rec - h_1) * inverse(r_1, order)) % order
```

The flag comes out to be: cs406{n0nc3_5h0uld_b3_u53d_0nc3}

Challenge - 4

First of all we notice that in both variants, the only thing not known to us is the Private Key.

Now, in variant 1, we notice that in the sign function, the only use of Private Key is in calculation of s, which is already given to us by the oracle. Since the equation of r and h use only the information we know, we can calculate them. Then as we know s, r, h and q (constant), we can rearrange the equation for s calculation to get the private key as follows:

$$s = (r + h * priv_k)\%order$$

$$\implies priv_k = (s - r) * h^{-1}\%order$$

Once we know the private key, the variant has been broken completely and we can calculate the signature of any message we want.
Here note that we only needed 1 access to the signature oracle and the message can be chosen arbitrarily.

In variant 2, we notice that the only difference lies in the calculation of r, which now uses private key. Hence we can not directly calculate r as in previous example.

But we also notice that the calculation of r only uses the first half of the message, While calculation of h, and hence s, uses complete message. Hence if the first half of 2 messages is the same and rest is different, then they would have the same r, but would have different h and s.

If we have 2 such messages $msg1$ and $msg2$ which have the same first half of the message, then:

$$r_1 = r_2$$

$$s_1 = (r_1 + h_1 * priv_k)\%order$$

$$s_2 = (r_2 + h_2 * priv_k)\%order$$

$$\implies s_1 - s_2 = ((h_1 - h_2) * priv_k)\%order$$

$$\implies priv_k = (s_1 - s_2) * (h_1 - h_2)^{-1}\%order$$

Once we know the private key, the variant has been broken completely and we can calculate the signature of any message we want.
Here note that we only needed 2 accesses to the signature oracle and those two messages must have the same first half.

The code written is as follows:

```
# -----VARIANT 1-----
msgs = ["a"] * 5

G = ecdsa.NIST256p.generator

msg = msgs[0].encode()
R = sigs[0][0]
s = sigs[0][1]
q = G.order()

r = int(hashlib.sha256(msg + str(VARIANT1_PUBKEY.x()).encode()).hexdigest(), base=16)
    % q
h = int(hashlib.sha256(str(R.x()).encode() + str(VARIANT1_PUBKEY.x()).encode() + msg)
    .hexdigest(), base=16) % q
```

```python

VARIANT1_PRIVKEY = ((s - r) * inverse(h, q)) % q

msg = challenge_msg_1

r = int(hashlib.sha256(msg + str(VARIANT1_PUBKEY.x()).encode()).hexdigest(), base=16)
     % q
R = r * G
h = int(hashlib.sha256(str(R.x()).encode() + str(VARIANT1_PUBKEY.x()).encode() + msg)
    .hexdigest(), base=16) % q
s = (r + h * VARIANT1_PRIVKEY) % q
# --------END--------

# -----VARIANT 2-----
msgs = ["ab", "ac", "a", "a", "a"]

G = ecdsa.NIST256p.generator

msg1 = msgs[0].encode()
R1 = sigs[0][0]
s1 = sigs[0][1]

msg2 = msgs[1].encode()
R2 = sigs[1][0]
s2 = sigs[1][1]

q = G.order()

h1 = int(hashlib.sha256(str(R1.x()).encode() + str(VARIANT2_PUBKEY.x()).encode() +
    msg1).hexdigest(), base=16) % q
h2 = int(hashlib.sha256(str(R2.x()).encode() + str(VARIANT2_PUBKEY.x()).encode() +
    msg2).hexdigest(), base=16) % q

VARIANT2_PRIVKEY = ((s1 - s2) * inverse(h1 - h2, q)) % q

msg = challenge_msg_2

r = int(hashlib.sha256(msg[:len(msg)//2] + str(VARIANT2_PRIVKEY).encode()).hexdigest
    (), base=16) % q
R = r * G
h = int(hashlib.sha256(str(R.x()).encode() + str(VARIANT2_PUBKEY.x()).encode() + msg)
    .hexdigest(), base=16) % q
s = (r + h * VARIANT2_PRIVKEY) % q
# --------END--------
```

The flag comes out to be: cs406{3dd54_g0t_m3_t4lk1ng_n0nc353nc3}