ASCS 2020 Qualifikation Schüler
Challenge write-up: IDBased1

Philipp Schweinzer

July 30, 2020

1 Challenge description

You are a new employee in a new company and you figure out that they use the Boneh-Franklin BasicIdent ID-based encryption scheme with the Weil pairing to encrypt the emails. Searching on the internal system, you recovered a partial sage implementation of the system.

The message are encrypted with the email address of the recipient as a public key. In particular, while sniffing the network, you found an interesting email sent to CEO@company.ch that you want to decrypt.

You also recovered some ciphertexts used to test the system. Fortunately, these test messages used all the CEO's public key.

You know that the plaintexts that were send are "This is the test message number" followed by a number (e.g. 5). Unfortunately, the ciphertexts are not in order...

1.1 Goal

Decrypt the message sent to the CEO.

2 Boneh-Franklin Identity-based encryption

The basis of this challenge is an encryption scheme called Boneh-Franklin Identity-based encryption. To try to solve this challenge, the first step is to look and understand this scheme. It is grouped into 4 different sections:

2.1 Setup

The PKG[1] chooses several different parameters:

- two groups G_1 and G_2 with size q
- a corresponding pairing e
- a randomly chosen private master-key $K_m = s \in \mathbb{Z}_q^*$
- a public key $K_{pub} = sP$
- a public hash function $H_1 : \{0,1\}^* \to G_1^*$
- a public hash function $H_2 : G_2 \to \{0,1\}^n$ for some fixed n
- the message and cipher space $\mathcal{M} = \{0,1\}^n,\ \mathcal{C} = G_1^* \times \{0,1\}^n$

2.2 Extraction

To create the public key for $ID \in \{0,1\}^*$, the PKG computes

- $Q_{ID} = H_1(ID)$
- the private key $d_{ID} = sQ_{ID}$ which is given to the user

2.3 Encryption

Given $m \in \mathcal{M}$, the ciphertext c is obtained as follows:

1. choose a random $r \in \mathbb{Z}_q^*$
2. compute $g_{ID} = e(Q_{ID}, K_{pub}) \in G_2$
3. set $c = (rP, m \oplus H_2(g_{ID}^r))$

[1] public key generator

2.4 Decryption

Given $c = (u, v) \in \mathcal{C}$, the plaintext can be retrieved using the private key:

$m = v \oplus H_2(e(d_{ID}, u))$

3 Failed attempts

The first step I had taken was to take a look at the sage implementation. After many hours of coding, I managed to complete the partial implementation into a working encryption system. Sadly this turned out to be an unnecessary effort.

My next thought was to brute-force the value of s, which would yield the master-key. Since K_{pub} and P were given I thought it would be easy to calculate s, because $K_{pub} = sP$. Looking back this was a silly idea, as the hardness of this calculation is based upon the ECDLP[2] and thus the whole idea of using elliptic curves in the first place.

4 Solution

4.1 Theory

After many hours of failed attempts, I took a look at the given ciphertexts with the "known" plaintext. This was when I noticed a similarity between two of these ciphertexts and the unknown ciphertext.
Remember $c = (u, v) = (rP, m \oplus H_2(g_{ID}^r))$. I noticed that u was the same for all three ciphertexts. This meant, that the random parameter r was also the same. Using this knowledge and given that we know the plaintext of one message, it is possible to retrieve the unknown message:

1. Let c_1 = ciphertext with known message m_1 and random number r_1.
2. Let c_2 = ciphertext with unknown message m_2 and random number r_2.
3. Assume that $r = r_1 = r_2$.
4. As $c_* = (u_*, v_*) = (rP, m_* \oplus H_2(g_{ID}^r)) \implies v_1 = m_1 \oplus H_2(g_{ID}^r)$

[2] elliptic curve discrete logarithm problem

5. Since m_1 is known to be "This is the test message number x", $x \in \mathbb{N}$
6. it is possible to extracte $H_2(g_{ID}^r) = v_1 \oplus m_1$.
7. Knowing this variable makes it possible to decypher the wanted message: $m_2 = v_2 \oplus H_2(g_{ID}^r)$

This solution is only possible, because these messages had the same random parameter r.

4.2 Implementation

Now it is time to implement this algorithm into sage and retrieve the flag. It is a very simple script, because it only needs to do two XOR operations:

```
import base64

def xor(xs, ys):
  return "".join(chr(ord(x).__xor__(ord(y))) for x, y in zip(xs,
      ys))

v1 = base64.b64decode('7ZP/X9jSV7SXdzPyJHlvuhu7AHOW8/A0UTUnlUL+
    URbc')
v2 = base64.b64decode('4LXZeMmDX9bXWxTmFF4oimniK0Sq39kURG4v')
m1 = 'This is the test message number 1'

h2 = xor(v1,m1)
m2 = xor(v2, h2)

print(m2)
```

4.3 Flag

The flag which was evaluated using the sage script:

YNOT18{B4DB4DB4DR4NDOMNE55}