Mandatory assignment 1

Thomas Hoffmann Kilbak (thhk)

01/03/2022

The problem

The problem with having a small `e` like 7, and a big modulo n, such as one that is 4096 bits long, it means that `m^e` is less than n, in which case calculating `m^e % n` will be the same as `m^e`.

This means that to get the original message, we can simply do the reverse operation on the ciphertext to retrieve the original message. That is `c^(1/e)`.

The problem is therefor a combination of the fact that a small e has been chosen, and also a very big n.

Step-by-step solution

1. Identify that n is longer than c, or that n is longer than `c^(1/e)`.
2. Calculate `a = c^(1/e)`.
3. Convert the number `a` to a byte array with big endian ordering.
4. Decode these bytes to text, yielding `HKN{...}`

Code to solve

Calculating the 7th root is trivial with small numbers, but to do so for big numbers where accuracy is also required, I can binary search from 0 to n and find the value `a` that satisfies `a^e = n`

A C# program that does this, could like this:

```
using System.Numerics;

class Program
{
    static void Main()
    {
        BigInteger n =
BigInteger.Parse("105012108016727244355604821445086736344105473244755760085
457432337102199334123341447178681954031163757852478231809070140147186465425
215483282902937451710923210623785516108045240380966713471097565607155249459
942450689166187723438368273481309525251634 1");
        int e = 7;

        BigInteger root = BigInteger.Pow(n, 1 / e);

        BigInteger high = n;
        BigInteger low = 0;

        while (high > low)
```

```
        {
            BigInteger mid = (high + low) / 2;
            BigInteger midPower = BigInteger.Pow(mid, e);
            if (midPower > n)
                high = mid;
            else if (midPower < n)
                low = mid;
            else
            {
                root = mid;
                break;
            }
        }
        var bytes = root.ToByteArray();
        Array.Reverse(bytes);
        var str = System.Text.Encoding.UTF8.GetString(bytes);
        System.Console.WriteLine(str);
    }
}
```

Other problems with RSA

I am a bit unsure what this part of the exercise is about. I have chosen to solve it, by researching other problems there can be when using a small public exponent.

One such problem is that, if a small e is chosen, then the number of possible d-values that satisfy $de \equiv 1 \pmod{\phi(n)}$ are lower. This makes it easier for an attacker to brute force all the possible d-values, and thereby get the private key.