Brute-Forcing a small CRT-RSA exponent

August 24, 2023

1 Introduction

In this challenge, we can ask the server to decrypt any RSA-cipher in two modes, either using the regular strategy or an optimized version leveraging the CRT-RSA. The difference in runtime is quite large, so the hint must be that there is something up with the CRT variant that we can exploit. (A few more details about CRT-RSA will follow once I have progressed more in my studying journey for this class. For now, suffice it to say that it involves computing a few auxiliary parameters in the RSA setup phase and using these auxiliary parameters for faster decryption).

2 The Brute-Force-Approach to Solving it

A bit of digging turns up a blog-post which describes a few apparently rather basic attacks on RSA. One of them is a brute-forcing attack on small CRT-RSA exponents. I will repeat what the author is saying there to better understand. But essentially, it's all Fermat's little theorem: if p is a prime and a is an integer co-prime to p, then $a^{p-1} \equiv 1 \pmod{p}$. This specific statement is a consequence of the theorem's general statement (which does not require p and a to be co-prime) $a^p \equiv a \pmod{p}$.

Two of the auxiliary parameters computed in the setup phase of CRT-RSA are d_p and d_q. They are chosen so that $d_p \equiv e^{-1} \pmod{p\text{-}1}$ and $d_q \equiv e^{-1} \pmod{q\text{-}1}$. In this attack, the assumption is that d_p or d_q is so small that it can be found with brute force. The key point of course is: how do we verify our guess? Looking at the definition of d_p, we can derive that $d_p * e \equiv 1 \pmod{p\text{-}1}$, or stated differently, $d_p * e = 1 + k * (p - 1)$. This is where Fermat's little theorem comes in!

Fermat's little theorem now tells us that for any number m co-prime to p (i.e., $gcd(m, p) = 1$), we have that

$$
\begin{aligned}
m^{e*d_p} &= m^{1+k*(p-1)} \\
&= m * m^{(p-1)^k} \\
&\equiv m * 1^k \\
&\equiv m (\text{mod } p)
\end{aligned}
$$

Consequently, $m^{e*d_p} \equiv m \pmod{p}$, i.e., $m - m^{e*d_p} = k * p$.

The approach thus consists in guessing values for d_p and $m < n$ and computing $gcd(m - m^{e*d_p}, n)$. Since $m - m^{e*d_p}$ is divisible by p if the guess is correct, the result will be p for a hit. See the blog post linked above for a code example. this YouTube video has another brute force approach, which I copied into brute-force.py.

3 The Efficient Exotic Approach to Solving it

The brute-force approach takes a long time. I was not sure it would work, and so I kept searching. I found this write-up, and this repo on GitHub. The approach detailed in these harnesses SageMath and does some magical multi-point evaluation of polynomial trees. Obviously I do not understand one bit, but I copied it and it worked in a rather speedy manner (apparently $O(min(\sqrt(d_p), \sqrt(d_q))))$! See the factor-sage.py script, and then crack.py for how I proceeded to grab the flag once N had been factored.