Performing a Haståd's Broadcast Attack in the *general* case.

August 29, 2023

1 Introduction

The collector challenge asked us to mount Haståd's Broadcast Attack in the simplest scenario where we have some small public exponent e (typically $e = 3$) and e ciphers corresponding to the same message encrypted with e and (generally) co-prime moduli n_i. In the general version of the attack, the messages are no longer identical, but polynomially transformed versions of each other.

Conveniently, I wrote a script for "harvesting" the required data from the server as it is a bit more cumbersome to come by than in the collector challenge. Here, the server responds to a single request with the name of a guest and their public key. The messages then are of the form "Dear GUEST flag{its_lit_l0l}". Since the messages thus differ based on the names of the guests, I figured it would be best to find three messages encrypted with public exponent $e = 3$ and with equally long guest-names. We'll see how this helps below. See collect.py for the collection part of the challenge.

2 Håstad's Broadcast - general version ft. Coppersmith

From the lecture slides: let's look at the theorem underlying the General Haståd broadcast, as well as its proof.

Thm. Let n_i be co-prime. Assume we modify some base message via $m_i = f_i(m)$, for $i \in \{1, ..., k\}$ and known polynomials f_i. If

$$k \geq e \times max\{deg(f_i) : i \in \{1, ..., k\}\},$$

then we can recover m from the f_i and $c_i = m_i^e \bmod n_i$. The corollary naturally is that any fixed padding scheme becomes dangerous, given enough messages. Use randomised padding. - Now the proof (also from the lecture slides)!

1. Put $g_i(x) = f_i(x)^e - c_i$, so all $g_i(m) \equiv 0 \bmod n_i$. Note: $deg(g_i) = e \cdot deg(f_i) \leq k$.
2. Using the Chinese Remainder Theorem, compute T_i so that $T_i \equiv 1 \bmod n_i$ and $T_i \equiv 0 \bmod n_j$ for $i \neq j$ and put

$$g(x) := \sum_{i=1}^{k} T_i \cdot g_i(x)$$

   adding degree $\leq k$, so $\deg(g) \leq k$
3. Now, $g(m) \equiv 0 \bmod n_i$ for all i:
   (a) summands $j \neq i$ vanish because of T_j
   (b) summand i vanishes by the definition of g_i
4. By CRT, $g(m) \equiv 0 \bmod \prod n_i$:

$$m < \min n_i < (\prod n_i)^{\frac{1}{k}} \leq (\prod n_i)^{\frac{1}{deg(g)}}$$

5. ... so we find m via Coppersmith.

Now, I should probably quickly say what Coppersmith is all about? Basically, Coppersmith showed that we can find all roots of a polynomial with degree e that lives in the mod n world in polynomial time. A root of a polynomial is a value for which the entire thing becomes zero (0 mod n in this case). The only condition that must hold for root x_0 to be found in polynomial time is that $|x_0| \leq \sqrt[e]{n}$.

So what do we do in our case? We know the general form of each plaintext ("Dear GUEST flag{its_lit_l0l}"), and we have collected 3 samples where $e = 3$ and the lengths of the three names are equal. We know how long the "Dear GUEST" part is, but obviously we do not know how long the flag is. Now the reasoning is this: for each of the ciphertexts, we know that the following condition holds: $c_i = (hex(\text{"}Dear\ GUEST\_NAME\text{"}) \cdot 2^{8 \cdot x} + hex(flag))^3$. Knowing this, we can apply the procedure given in the proof above, trying many values for x (essentially brute-forcing the length of the flag).

3 Disclaimer

Sadly, I did not capture the flag for this challenge! I did succeed however in cracking mocked data I generated myself (see general_hastad_test_prep.py and general_hastad_test.sage). I probably made some tiny mistake in the data collection of something and did not consider it worth my time to find it given that my method worked on a sanity check example.