CS2107 Assignment 1 Writeup

Easy Challenges

E.0

I used the Ctrl-F search function with the phrase “cs2107” to find the flag at the bottom of the document.

E.1 Rivest–Shamir–Adleman

The zip file contains the ciphertext, p, q and e and the name of the challenge also shows that the ciphertext was encrypted with RSA. I plugged in the values to an online RSA calculator (https://www.tausquared.net/src/pages/ctf/rsa) and was able to decrypt the message to plaintext. The online RSA decryptor uses the p, q and e to find the decryption exponent d using the equation:

d * e * mod (p-1)(q-1) =1

The plaintext was a string of numbers and after googling I realised it needed to be converted from decimal to hexadecimal first and then to ASCII text which I did so using an online converter tool (https://www.rapidtables.com/convert/number/hex-to-ascii.html).

E.2 xor_secure

Looking at the python source code, the plaintext has undergone XOR with key1, key2, key3, and key4. Since the order of XOR does not matter, I used an online XOR calculator (https://www.compscilib.com/calculate/binaryxor?variation=default) to XOR the encrypted message with all 4 keys. The final output is in hex which can be seen from the python source code as well. I then took the hex output and used an online hex to text converter (https://www.rapidtables.com/convert/number/hex-to-ascii.html) to obtain the flag.

E.3 Hash Browns

I used an online website which has a SHA1 hash database to perform a hash dictionary attack (https://hashes.com/en/decrypt/hash) on each of the hashes one by one which gave me the flag.

E.4 Caesar with a capital C

Looking at the source code, the Caesar cipher shifts alphabets and numbers to the corresponding values 5 positions away, wrapping around if it reaches the end of the number or alphabet list. I used an online website (https://www.dcode.fr/caesar-cipher) and keyed in 5 as the value to shift as well as the ciphertext to obtain the plaintext which was the flag.

Medium Challenges

M.2 Baby Shark

I opened the pcapng file in Wireshark, then exported all http objects. I found part 1 of the flag in the wallpaper.png file, part 2 of the flag in the inventory_book.xlsx file under the secret tab, part 3 of the flag in config.xml, and part 4 of the flag in the Confidential.pdf file in white font colour.

M.3 hash_key

Reading the source code, I realised that 2^25 key length is too short. Hence, I used a brute force algorithm to test all 2^25 keys until I obtained the flag.

Hard Challenge

H.1 Broadcasting

After a lot of googling, I realised that the the source code had a small exponent (e = 5) which would make it vulnerable to an attack called Hastad's broadcast attack (Small exponent attack), however many implementations of Hastad's attack will assume the plaintext being encrypted be the same for all parties. I later found out that the implementation of (a*flag + b) is considered a linear padding which still allows Hastad's attack to work. I have referenced the code from https://github.com/ValarDragon/CTF-Crypto/blob/master/RSA/hastadsSage.py

Hastad's attack works as follows:

We are given 5 ciphertexts, their corresponding moduli, a, and b.

The plaintext is encrypted with different private keys and moduli but the same encryption exponent.

Chinese Remainder Theorem (CRT) can then be used to construct a c such that:

$$c \equiv c_1 mod(N_1), c \equiv c_2 mod(N_2), \dots\ c \equiv c_5 mod(N_5)$$

$$where\ c_k\ is\ the\ kth\ ciphertext\ and\ N_k\ is\ the\ kth\ modulus$$

The Coppersmith method is then used to construct a 5-dimensional lattice since the polynomial is degree 5. The lattice is then reduced to find short lattice vectors that correspond to small roots. The small roots are then verified that they are valid.

After verification, there is one root remaining which is the plaintext.

The plaintext is then decoded to its original message.