CS641 - Assignment 6

Nishtha - 180489

Text on the wall :

“You see the following written on the panel:

n =

84364443735725034864402554533826279174703893439763343343863260342756678609216895093779263028809246505955647572176682669445270008816481771701417554768871285020442403001649254405058303439906229201909599348669565697534331652019516409514800265887388539283381053937433496994442146419682027649079704982600857517093

AlphaBeta: This door has RSA encryption with exponent 5 and the password is:

23701787746829110396789094907319830305538180376427283226295906585301889543996533410539381779684366880970896279018807100530176651625086988655210858554133345906272561027798171440923147960165094891980452757852685707020289384698322665347609905744582248157246932007978339129630067022987966706955482598869800151693”

Cryptanalysis

- **Commands to get to cipher**

  exit1 -> exit2 -> exit4 -> exit3 -> exit1 -> exit4 -> exit4 -> exit2 -> exit2 -> exit1 -> read

- **Analysis (RSA)**

  # **Figuring the commands** - The commands were quite tricky to figure out. Initially we entered with only 1 exit exit2 open and the exit1 from which we entered. Trying to input various exits we found that at each screen, only one out of 4 exits (exit5 was closed) caused a change in the hexadecimal codings at the end of screen and the other 3 took us to a exit we had already visited before. At each screen, we tried all 4 exits after disconnecting each time to finally get the cipher.

  # **Decryption** - The encryption and decryption functions for the RSA are given as follows:

Encryption : C = M^e mod N

Decryption : M = C^d mod N

We have been given C, N and e in our assignment and we have to figure out M. To find M, we will either have to factorise N or to find d both of which are not possible given large size of N. We confirm that padding has been used by checking that $$C^{\frac{1}{e}}$$ has not been integral. With given padding P, we have the equation as:

P + M = C^d mod N

We have been given a small e (=5), and we will exploit that to decrypt our cipher without finding d as in the special case in our slides. We will use the Coppersmith algorithm which is a low public exponent attack. It finds the small integer zeroes of univariate/bivariate polynomials modulo a given integer. The coppersmith theorem states that "Let N be an integer and f be a polynomial of degree δ. Given N and f, one can recover in polynomial time all x_0 such that

f(x_0) = 0 mod N and x_0 < N^{1/δ}.

Given this theorem, we have remodelled our problem as f(M) = (P + M)^e mod N. According to coppersmith, if M < N^{1/e}, then the password can be obtained as the root of the above polynomial.

We dont know the length of the password but we know that ASCII characters have been translated to password so it has to be a multiple of 8. Also, the length of password cant be greater than 200 bits from our assumption in the algorithm i.e. M < N^{\frac{1}{e}}. We converted our padding P to its binary form i.e. P_bin. Therefore, the final equation is polynomial = ((P_bin << length_M) + M)^e - C. Root if this polynomial has been obtained using the coppersmith and the Lattice reduction methods implemented in the code.

Figuring Padding - While figuring out the commands, we saw that only the numbers/characters at the end changes apart from the change in positions of 'twisted, narrow and rocky' in the message. The characters appeared to be hexadecimal and since it could be a message so we firstly tried to convert each pair (59, 6f, etc etc) to decimal form and find the corresponding ASCII character. Since these messages at each screen did not help in figuring out the exit, so we noted them anticipating that they might come handy while figuring out the padding in some way and so it happened. The messages found at each exit were:

EXIT	MESSAGE
exit1	You see
exit2	a Gold-B
exit4	ug in on
exit3	e corner

exit1 . It is
exit4 the key
exit4 to a tre
exit2 asure fo
exit2 und by
exit1

Combining this we got "You see a Gold-Bug in one corner. It is the key to a treasure found by" which we used as the padding in our first attempt and got it right.

References -

1. We studied about the Coppersmith theorem and Algorithm from the source - https://algo.epfl.ch/_media/en/projects/bachelor_semester/rapportetiennehelfer.pdf
2. For the code, we found an implementation here which was not exactly what we needed. We took help from it at certain stages to finally get our code right. It can be found here - https://github.com/mimoo/RSA-and-LLL-attacks/blob/master/coppersmith.sage

- **Password**

  B@hubAl!