CS641A Assignment-6

Sherlocked

Bholeshwar Khurana (170214) **Yaghyesh Chouhan (170813)**

Sarthak Dubey (180674)

1 How we cracked the cipher text:

We are given an RSA Encrypiton with:

N = 84364443735725034864402554533826279174703893439763343343863260342756678609216895093779263028809246505955647572176682669445270008816481771701417554768871285020442403001649254405058303439906229201909599348669565697534331652019516409514800265887388539283381053937433496994442146419682027649079704982600857517093

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

Cipher text, c = 58851190819355714547275899558441715663746139847246075619270745338657007055698378740637742775361768899700888858087050662614318305443064448898026503556757610342938490741361643696285051867260278567896991927351964557374977619644763633229896668511752432222528159214013173319855645351619393871433455550581741643299

To crack the ciphertext, we followed the following steps:

1. One of the obvious ways which can be used to decipher the password would be to factor N, which is impossible as N is too large. Other obvious approach would be to try finding d, for which we will require $\phi(N)$, and it cannot be computed without knowing the factors of N.
2. We can use **Coppersmith's algorithm** and **LLL Lattice Reduction Technique** to attack the RSA if the exponent e is small or the partial knowledge of secret key is available. As $e = 5$ is small, we can use this attack to decipher the password. **Coppersmith's algorithm** is as follows:

   > Let N be a positive integer and $f(x) \in \mathbb{Z}[x]$ be a monic, degree d polynomial. There is an algorithm that given n and f, efficiently, finds all integers x_0 such that $f(x_0) = 0 \mod N$ and $x_0 = N^{1/d-\epsilon}$, for some $\epsilon > 0$
   >
   > The algorithm runs in time $O(T_{LLL}(md, mlogN))$ where $m = O(k/d)$ for $k = min\{1/\epsilon, logN\}$

3. Let the ciphertext be c, modulus be N and exponent be e, we consider the possibility that a part of the deciphered message can be given to us in the form of padding. For this, we check whether $c^{1/e}$ is an integer or not.
4. It turns out that there is some padding p added to the original message m. So the final equation becomes:

   $$(p + m)^e = c \mod N$$

5. We formulate the polynomial as

$$f(x) = (p + x)^e - c \mod n$$

Here p is the padding (known to us), c is the ciphertext, e is the public key exponent and x is a Polynomial Ring of integers over modulo N. The root of $f(x)$ would be the original password.

6. We need to guess the padding p now. We looked at the problem statment which read *"This door has RSA encryption with exponent 5 and the password is "*. We assumed it to be the padding p and moved ahead.

7. We used the code *"RSA_break.sage"* to solve for x; the roots of $f(x)$. The code works as follows:
   - We translate padding p to its hex form *p_hex*.
   - Since the length of the password is unknown, but from our assumption $x_0 < N^{1/e} (\approx 10^{61})$; hence x_0 can't be longer than $\approx$ 200 bits.
   - Hence the final polynomial becomes $f(x) = ((p\_hex << length\_x) + x)^e - c \mod N$ and the polynomial ring is set over $\mathbb{Z}/n\mathbb{Z}$.
   - We iterate over the length of the password (length_x), i.e. 0 bits to 200 bits with an increment of 8 bits (1 byte) every time.
   - In every iteration the function "*solve(f(x))*" solves for the root(s) of $f(x)$ and if it finds a root, then the iteration is broken.
   - The function "*solve(f(x))*" uses SageMath's built-in function *small_roots()* [1] which finds the roots of f(x) which are less than $N^{1/e}$ using Coppersmith's algorithm.

8. On iterating over those lengths, we found the length of password to be 9 and the password to be **tkigrdrei**.

NOTE: To run the code *"RSA_break.sagews"*, you can visit https://www.cocalc.com and select SageWorksheet.

References:

[1] Sagemath Documentation: Here