CS641A **Team Gupt Nashak**
Modern Cryptology Roll Numbers: 14588,14746,14762
Prof. Manindra Agrawal sakshams, arareddy, tushant

Chapter 6: The Maze

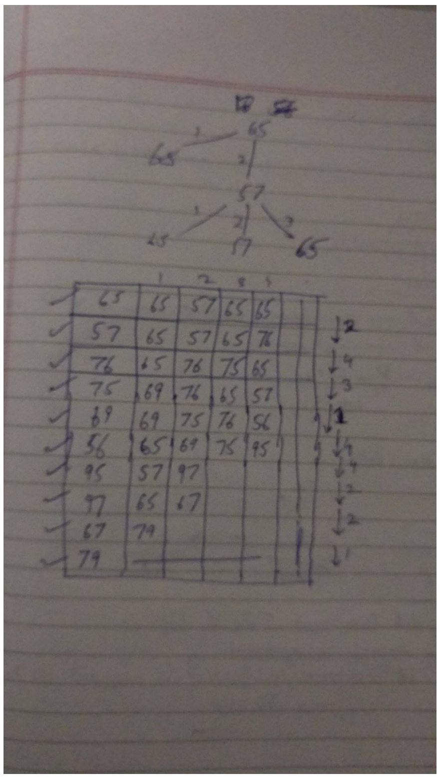
1 Prelude

1. *You are in a small chamber. Unlike the previous hall, this chamber seems naturally formed – the floor and walls are rocky and uneven. Was that hall and pool an illusion, you wonder. But then you notice a sign of human (or whatever) work: that someone has fixed lighted torches in the walls of the chamber, and these are providing ample light to see around. So you were not dreaming after all...*

You then notice that there are several exits from the chamber. You count five. And all but one of these exits have no doors! The exit with door has a panel next to it and the door is closed. Going closer to an open exit, you notice "Exit 2" written on the wall next to it. Curious, you go around and find that the exits are numbered from 1 to 5 in this fashion. The closed exit is numbered 5 and the exit from which you came in is numbered 1.

A — exit2,exit4,exit3,exit1,exit4,exit4,exit2,exit2,exit1

We moved through the maze by noticing that the symbols drawn on the screen are actually Japanese numerals and that they are change when we type different input from {exit1,exit2,exit3,exit4}. That allowed us to keep track of what state we are in. We made a transition table like one would do for a DFA (Deterministic Finite Automata). That allowed us to keep finding new states.


65
65
57
65
57
57
65
2
4
3
1
4
4
2
2
1

	1	2	3	4
65	65	57	65	65
57	65	57	65	76
76	65	76	75	65
75	69	76	65	57
69	69	75	76	56
56	65	69	75	95
95	57	97		
97	65	67		
67	79			
79				

2 Information

Finally, on the 10th state, when we typed *read*, we obtained the following text written on the plaque:

"n=90861646124264969076247659324816928011765416103817759776108942310270353
72288629337378949861774760127383275905459981741695802719615924314042014918832
38209100645198846429435542743866650618632237997136349770225731180040954362102
89594592747581643668331636012215542467748413499016657610311519500523204995393
504991

Gupt_Nashak: This door has RSA encryption with exponent 5 and the password is 5
14068444012778592135232041353837432200668066852186736312209046711599858889904
92674708587474294983572509101851682650858629740803994916477956408308036107041
37319116561663260654633687277328199777603448694114538080034406701534455323074
21911515872701255439838024407494016733302468850899667479450041684387602573 38"

3 Cryptanalysis:

We read through multiple attacks on RSA where the public exponent (e) is small. 2 such methods were Hastad's broadcast attack and Stereotyped messages attack by Coppersmith. Both of them relied on the Coppersmith method of finding small roots of polynomials using Lattice techniques, in conjunction with the LLL algorithm.

We concluded that the only feasible attack was if we knew the padding. In such a case, we could do a stereotyped message attack (where part of the message is known), and use it to find the actual message. All other attacks required information which was unfeasible (for instance, we need another encryption of the same message for the broadcast attack).

We first noted that n is 1024 bits long, and thus this must be RSA-1024. We went through a survey paper by Dan Boneh, titled: "Twenty Years of Attacks on the RSA Cryptosystem". This paper explained a method of implementing a lattice which lets us solve for a large number of message bits (100-150 approximately) if the padding is known.

We first attempted a simple solution where there was no padding (we assumed m must be small, of the order of 150 bits). This was not true, and we did not find such a solution.

So, we thought that maybe some other attack was possible in which the padding was not used. We thus tried, several attacks like ECM factorization hoping that it could factorize n, Fermat factorization in the hope that the primes were close and also Boneh Durfee's attack although it was unlikely that private exponent would be small as public exponent was already small.

We were then sure that we had to find the padding somehow.

Our initial attempts tried padding like *newline* repeated l times (where l is the padding length in bytes). Then we tried repeated occurrences of all ASCII characters by using a double loop.

The above mentioned paper includes the stereotyped messages attack, and uses an example padding "The password is ". We went through the complete level again, and noticed that we have a sentence which has "the password is" as a substring. The sentence is reproduced here:

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

What is more queer about this sentence is that it contains our team name, something that was never there in any of the problem statements up until level 5. We got suspicious and decided on trying this as a padding.

We had a few choices when decided how to use the padding. We converted the padding text to an integer (convert to binary and then to integer), and then tried to pad the space between the padding and the final text with 0s (assuming the total message was 1024 bits), something like this: `Padding.....00000....Message`. We then tried to pad in the following manner: `Padding.....Message....0000....`, totalling to 1024 bits.

None of the above revealed any information, and we finally tried the simplest padding scheme: `0000....Padding....Message....`. Implementing this was easy:

$$\texttt{Message = Padding * } 2^l \texttt{ + Password}.$$

Here, l is the length of the password. Since we did not know it, we had to loop over feasible values, from 1 to 150. Luckily, we found the solution this time. l was 104, and the value of password which satisfied the equation was: 5299087210244552546647101563485O5.

We converted it to binary, and then interpreted it as ascii using Sage's *bin_to_ascii* function, and obtained the following text: `cltFykfgTnlY`. It got submitted successfully.

4 Attributions

We obtained the mathematical background from the following paper:
https://crypto.stanford.edu/ dabo/papers/RSA-survey.pdf

We assumed LLL as a blackbox, as was done in class as well, and took help from the following repository on Github:
https://github.com/mimoo/RSA-and-LLL-attacks

Our final code involved using the functions provided in that repository, and using them to conduct a search for small roots of the formed polynomial.