Department of Electrical Engineering, IIT Bombay
EE 720 - An Introduction to Number Theory and Cryptography

Home Paper Report

Local Inverse method for breaking RSA based encryption-decryption

Prepared by:
Dishank Jindal (190070021) (Leader)
Divyanshi Kamra (19D070021)
Neeraj Nixon (20D070056)
Akshay Kaushal(20D070007)
Abhishek M(200070003)

Instructor: Prof. VR Sule
Date: January 6, 2023

Contents

1 Brief description of the work done

1.1 Overview

The home paper was read by all members of the group and then discussed for better understanding and clearer setting of goals for the implementation. We discussed the algorithm in detail and analysed corner cases. We then built the code in sagemath piece by piece in a modular format. The steps followed during the code implementation will be discussed in detail ahead in this report. We ensured that the key parameters to be chosen for the project to be kept random to avoid any sampling biases. After the code was built, 3 members of the project starting testing the code for smaller p,q,e values to ensure correctness. After this multiple simulations were carried out to analyse the algorithm and results have been compiled in a table in this report. The other 2 members started working on making this report for a concise representation of what we did in the project.

1.2 Theoretical Understanding

To define the maps consider two distinct primes p, q (both greater than 2), n = pq, e is a given number in [1, ϕ(n)) and d is a number such that ed = 1 mod ϕ(n). Two maps are defined as follows

$$\begin{aligned} &Encryption\ map\ y = E(x) \quad c = x^e\ mod\ n \\ &Decryption\ map\ y = D(x) \quad m = c^x\ mod\ n \end{aligned}$$

Here c is taken as m^e in the decryption setting. In the map E the choice of y is any value c in [1, n-1]. In the map D, y is obtained by choice of m in [1, n-1]. In this way a sample of multiple sequences S(F, y) can be constructed by choosing y for each of the functions. The sequences S(F, y) are defined for these functions as follows:

$$\begin{aligned} &\mathbf{F}\ y \quad S(F, y) \\ &\mathbf{E}\ c \quad \{c,\ c^e\ mod\ n,\ c^{e^2}\ mod\ n,\ c^{e^3}\ mod\ n...\} \\ &\mathbf{D}\ m \quad \{m,\ c^m mod n,\ c^{c^m\ mod\ n}\ mod\ n, ...\} \end{aligned}$$

Now, since we have got a sequence of numbers, calculating the minimal polynomial could be done by Berlekamp Massey algorithm. But another catch in this is that these are not in $\mathbb{F}_2$. To bring them into $\mathbb{F}_2$, we convert this sequence of scalars into a sequence of vector by doing a decimal to binary transform and thus a vector of length l nearly equal to log(n) is used instead of each scalar in the sequence S(f,y).

This problem converts to the one directly mentioned in the Home Paper and we present the algorithm used for the same. For an elements s_k of the sequence S(F,y) there is a vector $s_k = [s_k(1), s_k(2), ..., s_k(l)]^T$ where $s_k(i)$ is the i-th binary co-ordinate of s_k. Hence the sequence S(F,y) is a collection of l sequences for each of the i-th co-ordinates. Let these co-ordinate sequences be denoted as S(i). Then S(i) can be given as input to BM with field $\mathbb{F}_2$ and the LC and minimal polynomial of S(i) can be computed.

1. compute minimal polynomial $f_1(X)$ of $S(1)$.
2. Minimal polynomial $f \leftarrow f_1$
3. for $j = 2$ to l Verify: Boolean

$$b = f \text{ satisfies the recurrence relation for } S(j)$$

4. alternatively compute minimal polynomial of $S(j)$ by BM and verify whether $f_j | f$
5. If $b = 0$ compute minimal polynomial f_j of $S(j)$. (Alternatively if f_j does not divide f_i)
6. $f(X) \leftarrow \text{lcm}\,(f, f_j)$.
7. return Minimal polynomial of $\hat{s}$: $f(X)$
8. return LC of $S(F, y)$: $LC = \deg f(X)$.

Here, we compute the minimal polynomial of he scalar sequence using the Berlekamp Massey algorithm:

Algorithm using BM. Input: $\hat{s} = \{s_1, s_2, \ldots, s_{(2M-1)}\}$.

1. Compute minimal polynomial $f(X)$ of $\hat{s}$ using the BM function. (After using the scalar components of $\hat{s}$ as described below).
2. Compute x from (6).
3. If $y = F(x)$ is satisfied
4. Return local inverse x, $LC = \deg f$, $f(X)$.
5. EndIf
6. Else Return "insufficient data to compute local inverse".
7. End

This is used to compute the minimal polynomial for each of these sequences for a sample of values of y (after choosing an input value x) and computing the inverse x using the formula:

$$x = (1/a_0)[F^{m-1}(y) - (\sum_{i=1}^{(m-1)} a_i F^{(i-1)(y)})]$$

applied on the sequence ˆs of length l^2 part of S(F,y). The frequency of values y for which a correct inverse is found is the ratio, defined for specific functions E and D,

1.3 Code Implementation and Simulation

We tried to ensure that the code base remains highly modular for easy understanding as well as good scalability. We start by defining functions for the following:

- **func1:** Map E implementation
- **func2:** Map D implementation
- **func3:** Decimal to binary sequence conversion
- **func4:** $S(F, y)$ to $S(i)$ conversion
- **func5:** Computing Local Inverse of $S(F, y)$
- **func6:** Verifying x-y pair and updating $\nu(E)$
- **func7:** Verifying x-y pair and updating $\nu(D)$
- **func8:** Wrapper function for tabulating results for E
- **func9:** Wrapper function for tabulating results for D
- **func10:** Selecting random triplets (p, q, e) with given constraints

Now, we follow the steps given in the home paper assignment. We first call the **func10** to select a random triplet (p,q,e) such that p and q are prime numbers in the range from 8 to 10 bits long. n therefore is p*q. We then find out ϕ(n) as (p-1)*(q-1) and pick out a small e such that gcd(e, ϕ(n)) = 1. We now have a random triplet ready to be used. The modules in sage used for this are *random.sample*(), *prime_range*(), *gcd*().

We then call the wrapper function **func8** or **func9** depending on whether we are running the algorithm for encryption or decryption. The wrapper function then selects a random value in [1,n-1] as y and calls the **func1** or **func2** accordingly to implement the map and get the S(F,y). The modules in sage used for this are *power_mod*().

After this, wrapper function calls **func5** to compute the local inverse x if it exists for this chosen y. **func5** first calls **func3** to get the sequences S(i) and find the minimal polynomial using the BM method and subsequently computing LCMs. The final minimal polynomial is hence, used to compute the value x and is returned. The sage modules used in this part are *binary*(), *berlekamp_massey*(), *lcm*().

The returned value of x is now verified by using **func6** or **func7** accordingly. These function simply put the value of x in the forward maps E and D and check whether the image of x equals y. If yes, then the density function is updated, else the function simply returns. The sage modules used in this part are *power_mod*()

The wrapper code also calculates the time taken for one complete simulation for all the random samples of y. We start the inbuilt timer in sage and record the difference in time after the simulation is over and before we exit the function. The sage modules used for this part are *time.time*().

We finally return an array of compiled values of p,q,e,n,max_l,density, and time taken for the simulation to run. We run the simulation for 5 sample triplets of (p,q,e) and 10,000 values of y for each of E and D maps.

2 Results

2.1 Compiled Table

p	q	e	n	l	$n(S_y)$	$\nu(E)$	$Time(s)$
257	487	5	125159	10	10,000	0.0376	1241
587	971	3	569977	10	10,000	0.0021	835
311	431	3	134041	10	10,000	0.0303	848
389	487	5	189443	10	10,000	0.0274	540
263	431	3	113353	10	10,000	0.0086	395

p	q	e	n	l	$n(S_y)$	$\nu(E)$	$Time(s)$
1151	683	3	786133	12	10,000	0.0012	739
421	349	11	146929	10	10,000	0.0099	357
373	463	5	172699	10	10,000	.0089	440
281	421	11	118301	10	10,000	.0321	169
257	439	5	112823	10	10,000	.0069	290

2.2 Individual Notebook Snippets

2.2.1 Outputs for function E

```
table_parameters_E [587, 971, 3, 569977, 11, 10000, 0.0021, 835.7745809555054]
```

```
table_parameters_E [[587, 971, 3, 569977, 11, 10000, 0.0021, 835.7745809555054]]
```

```
table_parameters_E [257, 487, 5, 125159, 11, 10000, 0.0376, 1241.6791231632233]
```

```
table_parameters_E [[257, 487, 5, 125159, 11, 10000, 0.0376, 1241.6791231632233]]
```

```
table_parameters_E [311, 431, 3, 134041, 11, 10000, 0.0303, 848.551696062088]

table_parameters_E [[311, 431, 3, 134041, 11, 10000, 0.0303, 848.551696062088]]

  table_parameters_E [389, 487, 5, 189443, 11, 10000, 0.0274, 540.0756194591522]

  table_parameters_E [[389, 487, 5, 189443, 11, 10000, 0.0274, 540.0756194591522]]

table_parameters_E [263, 431, 3, 113353, 11, 10000, 0.0086, 395.62326669692993]

table_parameters_E [[263, 431, 3, 113353, 11, 10000, 0.0086, 395.62326669692993]]
```

2.2.2 Outputs for function D

```
table_parameters_D [1151, 683, 3, 786133, 11, 10000, 0.0012, 739.5914356708527]

table_parameters_D [[1151, 683, 3, 786133, 11, 10000, 0.0012, 739.5914356708527]]

  table_parameters_D [421, 349, 11, 146929, 11, 10000, 0.0099, 357.9420132637024]

  table_parameters_D [[421, 349, 11, 146929, 11, 10000, 0.0099, 357.9420132637024]]

  table_parameters_D [373, 463, 5, 172699, 11, 10000, 0.0089, 440.64019799232483]

  table_parameters_D [[373, 463, 5, 172699, 11, 10000, 0.0089, 440.64019799232483]]

   table_parameters_D [281, 421, 11, 118301, 11, 10000, 0.0321, 169.84551978111267]

   table_parameters_D [[281, 421, 11, 118301, 11, 10000, 0.0321, 169.84551978111267]]
```

```
table_parameters_D [257, 439, 5, 112823, 11, 10000, 0.0069, 290.3413565158844]
```

```
table_parameters_D [[257, 439, 5, 112823, 11, 10000, 0.0069, 290.3413565158844]]
```

3 Learning and Conclusion

Through this project we get an understanding and practical knowledge to implement Local Inverse method for breaking RSA based encryption-decryption. Doing this project gave a much better understanding of how RSA encryption works and why is it computationally very difficult to break.

We learned how to implement Berlekamp Massey algorithm to compute the minimal polynomial and how to compute inverse when minimal polynomial is known. We take this inverse and compare it with the correct inverse to find the density of the functions E and D for 5 different sets of {p, q, e}.

For the function E, from the 5 sets of values we used, it can be seen that we got the maximum matching for p = 257, q = 487, e = 5, and the ratio = 0.0376. Averaging all 5 values $v(E)_{avg} = 0.0212$

For the function D, from the 5 sets of values we used, it can be seen that we got the maximum matching for p = 281, q = 421, e = 11, and the ratio = 0.0321. Averaging all 5 values $v(D)_{avg} = 0.0118$

Using Local inverse method for breaking RSA encryption in this case is fairly simple to compute as the prime numbers are fairly small. In real-life these will be 1,024 bit prime numbers, and N will have 2,048 bit numbers, which will be extremely difficult to factorize.

4 References

1. Programming in Sage
2. Berlekamp-Massey algorithm to find the minimal polynomial of a linear recurrence sequence
3. Measuring execution time in Sage