BROADCASTING AND LOW EXPONENT RSA-ATTACK

Riya Suchdev

riya.suchdev@sjsu.edu

Tazmina Sharmin

tazminabd@gmail.com

Table of Contents

Riya Suchdev (riya.suchdev@sjsu.edu)
Tazmina Sharmin (tazminabd@gmail.com)

Abstract

The project[4] was aimed to deciphering a message where the same message was delivered to three different recipients encrypted by their corresponding provided public keys using **RSA**[1] algorithm. Fortunately the exponents used in all the encryptions were same.

Introduction

We know that in the practical RSA we use large primes, now-a-days 2048 bits. And also **PKCS**[5] is used to take measures so that direct computations are not possible to decrypt the message. The given problem is the strong demonstration of why we need the above.

Solution

Algorithm

Since the cipher texts and the public exponents are given, we can formulate the problem mathematically like below:

a1, a2, a3	The ciphertexts
p1, p2, p3	The public key from the three certificate files
msg	Be the original message
x	$x = msg^3$, since the exponent is 3
x = a1 mod p1 x = a2 mod p2 x = a3 mod p3	Based on the encryption

We can apply Chinese Remainder Theorem[2] on these conditions iff

- p1, p2 and p3 are relatively prime

And then we can do the following:

p12 = p1 * p2	p12′ = mod inverse of p12 w.r.t p3
P23 = p2 * p3	p23′ = mod inverse of p23 w.r.t p1
P13 = p1 * p3	p13′ = mod inverse of p13 w.r.t p2
M = p1 * p2 * p3	

Riya Suchdev (riya.suchdev@sjsu.edu)
Tazmina Sharmin (tazminabd@gmail.com)

And then we can find the msg by

```
x = (a1 * p23 * p23’ + a2 * p13 * p13’ + a3 * p12 * p12’) mod M
```

```
msg = cubic root of x
```

Now we can decrypt the message only if $msg^3 = x$. And then if the condition holds, we can simply break the `msg` into ascii characters.

Input

We are given three certificate files containing the public keys of the three receivers - Zert1.txt, Zert2.txt and Zert3.txt. Since the files have additional information, the extracted public keys are:

Zert1.txt	00:96:23:51:1e:67:69:64:4d:69:3e:89:f6:92:ff:c2:55:8e:ef:12:1d:42:ca:98:69:97:81:e1:39:e2:9c:2e:1a:a5:8d:88:83:bb:db:a4:11:65:fd:eb:85:a9:a5:64:8f:c2:9a:65:d5:9e:94:01:69:4d:d1:1a:e2:05:f0:ce:3b
p1	78633628283969454226716416510929008687874183044165827348065545984660288831079698397320757101009209110739680482651971525454048174982662148597966531839135 31
Zert2.txt	00:ad:4b:c0:f9:80:f4:52:3f:49:0f:c4:0c:12:ef:ce:cc:1e:8a:f6:78:90:b6:56:24:49:87:6e:8e:09:1e:86:1c:da:69:9e:5a:8e:b3:09:b0:a9:d6:b2:93:10:0c:12:29:fb:d1:8a:59:51:f3:3b:6f:ba:b1:fd:8d:90:f7:c8:29
p2	907624344020368032154223860993777467933763152168118722850190327867160748848153790265696206875867273251338612843853371141752857158125392535072479110127824 9
Zert3.txt	00:b7:22:33:64:d8:83:53:ec:02:b0:85:0e:8a:01:d2:ba:9c:a2:66:3c:32:c1:5d:f7:b5:96:40:6c:6f:c1:c1:71:ac:96:5a:55:4b:8b:33:8f:4b:b0:46:c5:43:93:7b:4b:19:c6:99:86:4f:1d:0d:d4:be:01:77:ec:cc:e0:bb:57
p3	959148472732584167625134311317612125364389819933875614838530210517188852365792510831724039103037540204147255833957997224802567276316098760105426411537903 1

We are also given

a1	276662577646849051702044216520062258535397236473425749363465628542884981808843667974301776828895128554214939622035541208847872984310140455659506041906353 0
a2	324012680060397741295489485035658649977822349740154313577170439118735275123018558107274044009323333917121974448936522235186584449171431259660556273615539 2
a3	782657205015069626620909175768899543205346321163114020666437791518563955029983787311935279071951212072600213849293562476809725166534434695067385887696520 4

Riya Suchdev (riya.suchdev@sjsu.edu)
Tazmina Sharmin (tazminabd@gmail.com)

Output

We calculated:

p12	71369795289179233877852691498378278930747462958017173290224169237290817441169713215864684199327224651252235417067626231056097127118068194455885964129593966783607552487519583420369912082725132836081062458333483152280536371184579496010534882004855241327458456314192166212741132459926021644718252574058387087219
p23	87054650338204955949884054972152559929827396748343975292726560114356965203846586690929925326857704175854908861216884437887397606297170715756686223539029568253925651847194163559700612823778030891515268788637584080174932383168539934468543327300471347469875207400702302796615812078700322317271956846643230996719
p13	75421324473991035239984973615601156035782562091931233826035510348835740322761556308445346919312192995474080448384435792810560362461644500707887294796791513885554146812175512702924424684467806517303139190945099423213330337680774918952948284232823090467625679172229365971417973602715460296639716249191994568461

p12'	84255635212408027434813322930393101402182551170902961036261650982383946955610471245556027051406228696597422297516907170840020008788433356905541304392978 29
p23'	70909698988582368647245487386009768544880819643802292488640882387582242852073250287185153519486818370313181027115380663947397757837366299343025811765671 24
p13'	19948198371692535425928745337919818848289158475382652344525375025128608812688510757801906086583232573359944754641942788133151543078358445199586613985619 52

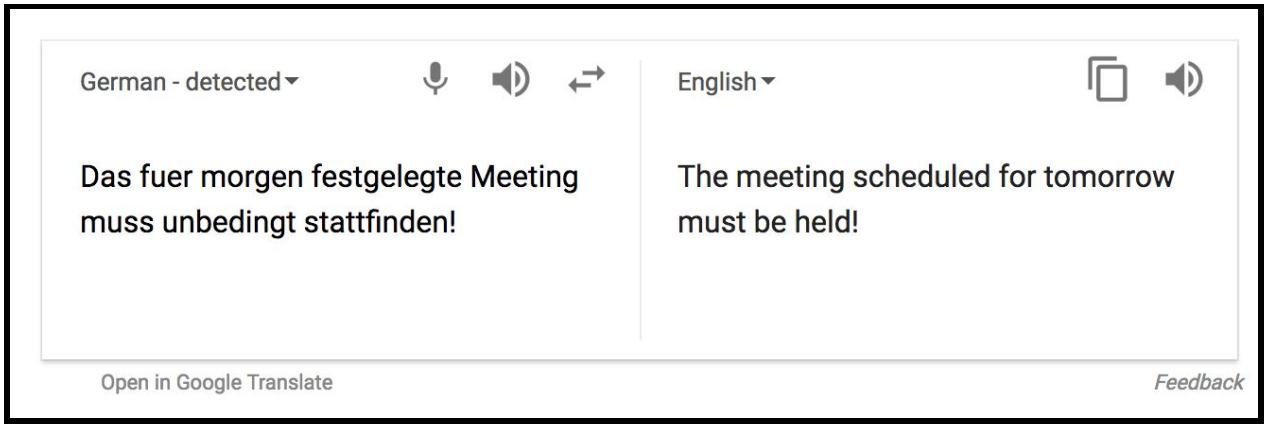
M	68454230150853442383572644682841828958820589526726684697939786950116251689837369584263384319249843207443634213613056435253723420813844857248501483169399686120075877524157117486003876475606917622377085466537053956371812505267154986625802928222291240629793943020223696601965893229807639001726391302650377563343410632943197952535914660252994617839225802609062560876393137668352228433890054638861812272996491007159899900245192492841386859749471889663229136224070478 9
x	27380001572369348143380930054422156143180482933760285201131369276589044572352562359991444352617412433062971760662580475962048334685399998562459425255534710795445833709496963043078944070333249743091538659752415253427856065061435620292451988745743154011240772786351417658799187161917627453116051032210090691030558159156738312880297059228636608197688341715607133715294036968153846444520789086607600295865689792450010525842590071602371577777625104859083136 97
msg	13989788739742427272277193015453900250111512583793201009668234415576204048672760581156999690701624926694924550547324254392117134178836228545125433044513

After decoding the message we got:

```
Das fuer morgen festgelegte Meeting muss unbedingt stattfinden!
```

Riya Suchdev (riya.suchdev@sjsu.edu)
Tazmina Sharmin (tazminabd@gmail.com)

The meaning of the message is (courtesy of google translate):



Verification

gcd(p1, p2)	1	Relatively prime
gcd(p2, p3)	1	Relatively prime
gcd(p1, p3)	1	Relatively prime
msg^3	2738000157236934814338093005442215614318048293376028520113136927658904457235256235999144435261741243306297176066258047596204833468539999856245942525553471079544583370949696304307894407033324974309153865975241525342785606506143562029245198874574315401124077278635141765879918716191762745311605103221009069103055815915673831288029705922863660819768834171560713371529403696815384644452078908660760029586568979245001052584259007160237157777762510485908313697	Same as **x**

Riya Suchdev (riya.suchdev@sjsu.edu)
Tazmina Sharmin (tazminabd@gmail.com)

Screenshots

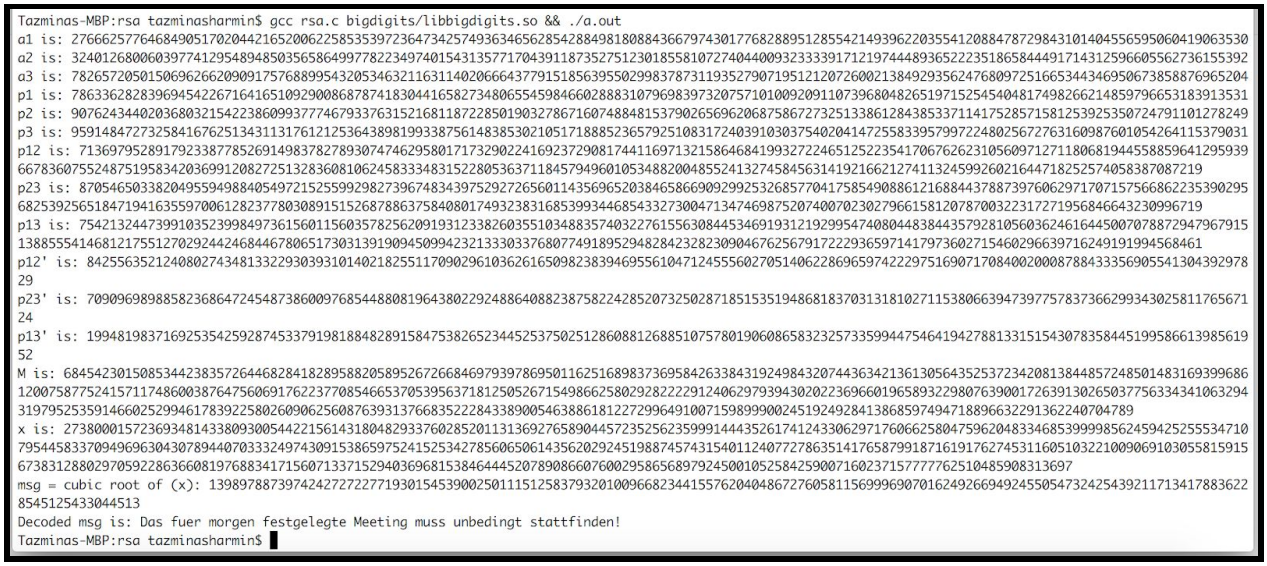


Fig. 1: Code Output

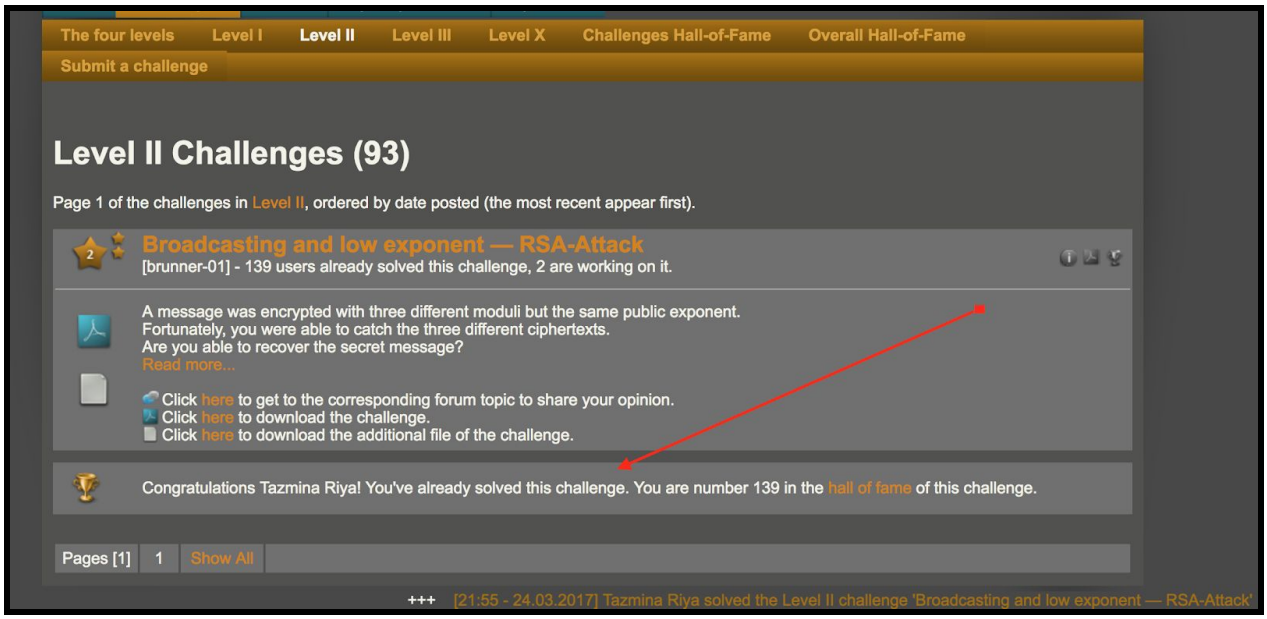


Fig. 2: Submission

A Harder Problem

A harder problem of this version could be designed like this:

> Given the same problem with same constraints except that only one of the receivers is using a different exponent.

This is a slightly harder version of the problem, but still could be solved by first finding $\mathbf{x}$ for all the ones with same exponent. And the other one can be found by sieving. Though it might take some time to get the message which matches $\mathbf{m^e}$.

Preventive measures

There are various methods to prevent this attack on RSA.

- PKCS standards to padding random bits to the message before encryption. These numbers will add as buffer so that the ciphertext will not be easily broken.
- The other way is to have a large e. A popular encryption component value for e is $2^{16}+1$. The main advantage is that the same message has to be sent to $2^{16}+1$ people before Chinese Remainder attack to be successful.

Conclusion

We have successfully decrypted the RSA with low exponent value using Chinese Remainder theorem. We have also discussed the preventive measures and mentioned a harder version of the problem. We also needed to use a non standard **C** biginteger library[3] since **C** doesn't have any standard biginteger library.

Bibliography

1. RSA (cryptosystem), https://en.wikipedia.org/wiki/RSA_(cryptosystem), Accessed on March 10, 2017

2. Chinese Remainder Theorem, https://en.wikipedia.org/wiki/Chinese_remainder_theorem, Accessed on March 10, 2017

3. BigDigits multiple-precision arithmetic, http://www.di-mgt.com.au/bigdigits.html, Accessed on March 14, 2017

4. Broadcasting and Low Exponent RSA Attack, https://www.mysterytwisterc3.org/en/challenges/level-ii/broadcasting-and-low-exponent-rsa-attack, Accessed on March 15, 2017

5. PKCS 1, https://en.wikipedia.org/wiki/PKCS_1, Accessed on March 15, 2017