

CS 265 – CRYPTOGRAPHY AND COMPUTER SECURITY
BROADCASTING AND LOW EXPONENT- RSA ATTACK

Submitted By

SAMYUKTHA SRIDHARAN (010030371)

TINA PHILIP (010019958)

November 7, 2014

Submitted To

Dr. Mark Stamp

Department of Computer Science

San Jose State University

Samyuktha Sridharan (email.samyu@gmail.com)

Tina Philip (talk2tinaphilip@gmail.com)

TABLE OF CONTENTS:	PAGE NO.
1. INTRODUCTION	3
2. LITERATURE REVIEW	3
3. ALGORITHM DESIGN	4
4. IMPLEMENTATION DETAILS	4
5. RESULT	5
6. ENHANCEMENTS	6
7. SOLUTION TO ATTACKS	8
8. FUTURE WORKS	8
9. CONCLUSION	8
10. REFERENCES	8

TABLE OF FIGURES:	PAGE NO.
1. Implementation of bigdigits library	4
2. Screenshot of Broadcast Attack output	6
3. Screenshot success of Broadcast Attack	7
4. Screenshot of Alice's Birthday Attack output	7
5. Screenshot success of Alice's Birthday Attack	7

ABSTRACT

The aim of the project is to study RSA algorithm and possible attacks on the algorithm in detail. The project deals with solving a mathematical attack called '*Broadcast Low Public Exponent*' attack on RSA cryptosystem by implementing Chinese Remainder Theorem. The implementation was successful by getting the plain text as the output.

1. INTRODUCTION

RSA algorithm has been proposed and studied for over three decades. RSA algorithm is based on a hard factoring problem. Extensive research is being carried out in attacking the RSA crypto system and there is no efficient algorithm (till date) to the factoring problem in a reasonable amount of time, if RSA is implemented correctly. However, there are some known attacks exploiting the naïve implementations of RSA. In this project, the same message which is broadcast to three different parties is decrypted by implementing Chinese Remainder Theorem. Given three different cipher text, and three different large public parameters, and a small public exponent, the message is decrypted efficiently. The implementation is written in C programming language.

2. LITERATURE REVIEW

RSA algorithm [1] is one of the first known practical approaches to public key cryptosystems. Invented by Ron Rivest, Adi Shamir and Leonard Adleman in 1977, this algorithm is based on a hard mathematical problem, called factoring problem. RSA algorithm is widely used, till date due to the fact that factoring problem is very hard.

In a simple RSA algorithm, we have the public key (N, e) which is known to everyone and d is the private key, which is known only to self. N is computed as the product of two large prime numbers p and q . Also e and d are two integers, satisfying $e \cdot d \equiv 1 \pmod{\phi(N)}$ where $\phi(N) = (p-1) \cdot (q-1)$ the Euler's totient function of N [2].

There have been many known attacks for RSA exploiting the parameters used during the construction of the RSA cryptosystem. The attacks are broadly classified into two categories namely, mathematical attacks which focus on the underlying structure of RSA function, and implementation attacks, which focus on the pitfalls during implementation [3].

Factoring of public component N is not involved in mathematical attacks. Mathematical attacks can be classified as Small Private Key attacks and Small Public key attacks. The small private key is chosen to increase the performance of decryption during run time. But this is subject to attack proposed by M. Wiener [4] proved that with a small decryption it is efficient to crack the RSA cryptosystem.

On the other hand, when using a small public exponent, it may be possible to perform a simple cube-root attack. In the case where the same message M is broadcast to three different users' public modulus and $e=3$, given cipher texts C_1, C_2, C_3 , eavesdropper (Trudy) will be able to decrypt the plaintext M , without having to factorize to obtain d . This is done using Chinese Remainder Theorem [9] and is called the '*Hastad Broadcasting attack*' [2]. This project aims at implementing the Hastad Broadcast attack in an efficient manner.

3. ALGORITHM DESIGN

The implementation is done by adopting the following algorithm:

Step 1: The public exponents are given to be $e=3$; n_1, n_2, n_3 .

Step 2: From the definition of RSA algorithm, it is known that

$$c_1 = m^3 \bmod n_1; c_2 = m^3 \bmod n_2; c_3 = m^3 \bmod n_3$$

[Since there are 3 messages encrypted using n_1, n_2, n_3 , as given in the problem description]

Now verify if n_1, n_2, n_3 are pair wise co prime. $\text{GCD}(n_1, n_2)=1, \text{GCD}(n_2, n_3)=1; \text{GCD}(n_3, n_1)=1$

Step 3: Compute $N = n_1 * n_2 * n_3$

Step 4: Compute $N_1 = N/n_1; N_2 = N/n_2; N_3 = N/n_3$

Step 5: Compute $d_1 = N_1^{-1} \bmod n_1, d_2 = N_2^{-1} \bmod n_2, d_3 = N_3^{-1} \bmod n_3$

Step 6: Compute $M = (c_1 * N_1 * d_1 + c_2 * N_2 * d_2 + c_3 * N_3 * d_3) \bmod N$.

Step 7: Compute cube root of M to obtain plain text message = $M^{1/3}$

4. IMPLEMENTATION DETAILS

The implementation of this project is done using C programming language. C programming allows handling of long integers, which has the capability of storing up to 64 bits. However, in the problem statement, the cipher text and the public component 'n' consists of 512 bits. Hence, leveraging the use of bigdigits library [6] seemed feasible in this scenario. The 'bigdigits' library provides implementation for various arithmetic routines, such as modulo, multiplication, etc. to carry out operations for very large numbers. A snapshot of one such function is given below:

```
int bdMultiply(T w, T u, T v)
/* Compute w = u * v
   -- no overlap permitted
*/
{
    size_t dig_size;

    assert(w && u && v);
    /* Check for cheaper option */
    if (v->ndigits == 1)
        return bdShortMult(w, u, v->digits[0]);

    /* Make sure u and v are the same size */
    dig_size = max(u->ndigits, v->ndigits);
    bd_resize(v, dig_size);
    bd_resize(u, dig_size);
    /* Now make sure w is big enough for product */
    bd_resize(w, 2 * dig_size);

    /* Finally, do the business */
    mpMultiply(w->digits, u->digits, v->digits, dig_size);

    /* Make sure we've set the right size for w */
    w->ndigits = mpSizeof(w->digits, 2 * dig_size);

    return 0;
}
```

Figure: 1 Implementation of bigdigits library [6]

5. RESULT

The results obtained after implementing the broadcast attack is given in Figure 4. The results include the hex value and the plain text format with spaces and special characters.

Once the result obtained in command prompt, it was verified against the Crypto Challenge. The verification is shown in Figure 5.

```
-----Obtaining PlainText from Broadcast Low Public Exponent Attack-----
```

```
The public modulus values
```

```
n1=78633628283969454226716416510929008687874183044165827348065545984660288831
07969839732075710100920911073968048265197152545404817498266214859796653183913
531
```

```
n2=90762434402036803215422386099377746793376315216811872285019032786716074884
81537902656962068758672732513386128438533711417528571581253925350724791101278
249
```

```
n3=95914847273258416762513431131761212536438981993387561483853021051718885236
57925108317240391030375402041472558339579972248025672763160987601054264115379
031
```

```
Low public exponent e= 3
```

```
The ciphertexts are:
```

```
c1=34d2fc2fa4785e1cdb1c09c9a5db98317d702aaedd2759d96e8938f740bf982e2a42b904e5
4dce016575142f1b0ed112cc214fa8378b0d5eebc036dc7df3eeea
```

```
c2=3ddd68eeff8be9fee7d667c3c0ef21ec0d56cefab0fa10199c933cfffbf0924d486296c604a
447f48b9f30905ee49dd7ceef8fc689a1c4c263c1b3a9505091b00
```

```
c3=956f7cbf2c9da7563365827aba8c66dc83c9fb77cf7ed0ca225e7d155d2f573d6bd18e1c18
044cb14c59b52d3d1f6c38d8941a1d58942ed7f13a52cacc48154
```

```
gcd(n1,n2)=1
```

```
gcd(n2,n3)=1
```

```
gcd(n3,n1)=1
```

```
n1,n2 and n3 are pairwise coprime
```

```
N=48b497d0a83ed41f3516cbf4d62720c8c7322f361ddd9363efd298669d226043f0e80c30a8e
54ddf63f3d1922821e7d06df80437aca0dfdee9e7edf4f0019010e4116619cb43552d4d34a529
af1766cd829e4713
```

```
a5f118ffc31502fc630eabb49d6fa206a9d68e74b02cbabb9e3b772b0ca8d88032205466f9f95
dd1a7b115
```

```
Message =
```

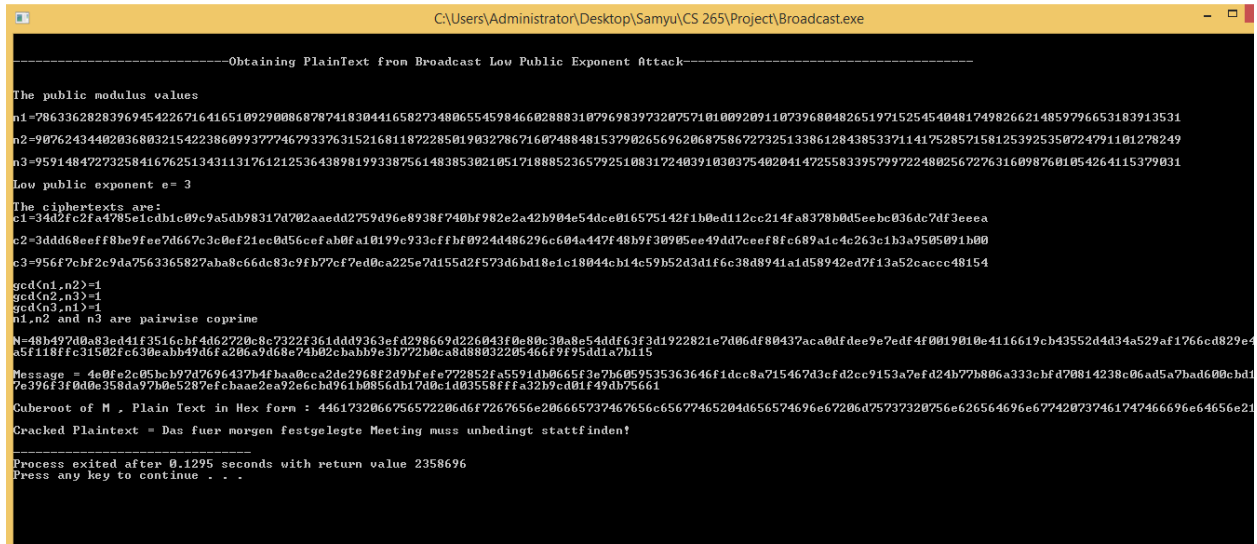
```
4e0fe2c05bcb97d7696437b4fbaa0cca2de2968f2d9bfefef772852fa5591db0665f3e7b605953
5363646f1dcc8a715467d3cfd2cc9153a7efd24b77b806a333cbfd70814238c06ad5a7bad600c
bd1b807e396f3f0d0e358da97b0e5287efcbaae2ea92e6cbd961b0856db17d0c1d03558fffa32
b9cd01f49db75661
```

Broadcasting And Low Exponent – RSA Attack

Cuberooot of M , Plain Text in Hex form :

```
4461732066756572206d6f7267656e206665737467656c65677465204d656574696e67206d75737320756e626564696e677420737461747466696e64656e21
```

Cracked Plaintext = Das fuer morgen festgelegte Meeting muss unbedingt stattfinden!

A screenshot of a Windows command prompt window titled "C:\Users\Administrator\Desktop\Samy\CS 265\Project\Broadcast.exe". The output shows the process of obtaining plaintext from a broadcast low public exponent attack. It lists three public modulus values (n1, n2, n3) and a low public exponent e=3. It then shows the ciphertexts for each modulus. The process involves calculating the cube root of the product of the ciphertexts to recover the plaintext. The final output shows the cracked plaintext in hex and its ASCII representation: "Das fuer morgen festgelegte Meeting muss unbedingt stattfinden!".

```
C:\Users\Administrator\Desktop\Samy\CS 265\Project\Broadcast.exe

-----Obtaining PlainText from Broadcast Low Public Exponent Attack-----

The public modulus values
n1=7863362828396945422671641651092900868787418304416582734806554598466028883107969839732075710100920911073968048265197152545404817498266214859796653183913531
n2=907624344028368032154223860937774679337631521681187228501903278671607488481537902656962068758672732513386128438533711417528571581253925350724791101278249
n3=9591484727325841676251343113176121253643898199338756148385302105171888523657925108317240391030375402041472558339579972248025672763160987601054264115379031
Low public exponent e= 3

The ciphertexts are:
c1=34d2fc2fa4785e1cdbc09c9a5db98317d702aaedd2759d96e8938f740bf982e2a42b904e54dce016575142f1b0ed112cc214fa8378b0d5eebc036dc7df3eeea
c2=3ddd68eeff8be9fee7d667c3c0ef21ec0d56cefab0fa10199c933cfff0924d486296c604a447f48b9f30905ee49dd7ceef8fc689a1c4c263c1b3a9505091b00
c3=956f7cbf2c9da7563365827aba8c66dc83c9fb77cf7ed0ca225e7d155d2f573d6bd18e1c18044cb14c59b52d3d1f6c38d8941a1d58942ed7f13a52caccc48154

gcd(n1,n2)=1
gcd(n2,n3)=1
gcd(n3,n1)=1
n1,n2 and n3 are pairwise coprime

N=48b497d0a83ed41f3516cbf4d62720c8c7322f361ddd9363efd298669d226043f0e80c30a8e54ddff63f3d1922821e7d06df80437aca0dfdee9e7edf4f0019010e4116619cb43552d4d34a529af1766cd829e4af118ffc31502fc630eabb49d6fa206a9d68e74b02cbabb9e3b772b0ca8d88032205466f9f95dd1a7b115

Message = 4e0fe2c05bcb97d7696437b4fbaa0cca2de2968f2d9bfe7e722852fa5591db0665f3e7b6059535363646f1dccc8a715467d3cf42cc9153a7efd24b77b806a333cbfd70814238c06ad5a7bad600cbd17e396f3f0d0e358da97b0e5287efcbaae2ea92e6cbd961b0856db17d0c1d03558fffa32b9cd01f49db75661

Cuberooot of M . Plain Text in Hex form : 4461732066756572206d6f7267656e206665737467656c65677465204d656574696e67206d75737320756e626564696e677420737461747466696e64656e21
Cracked Plaintext = Das fuer morgen festgelegte Meeting muss unbedingt stattfinden!

Process exited after 0.1295 seconds with return value 2358696
Press any key to continue . . .
```

Figure 2: Screenshot of Broadcast Attack output

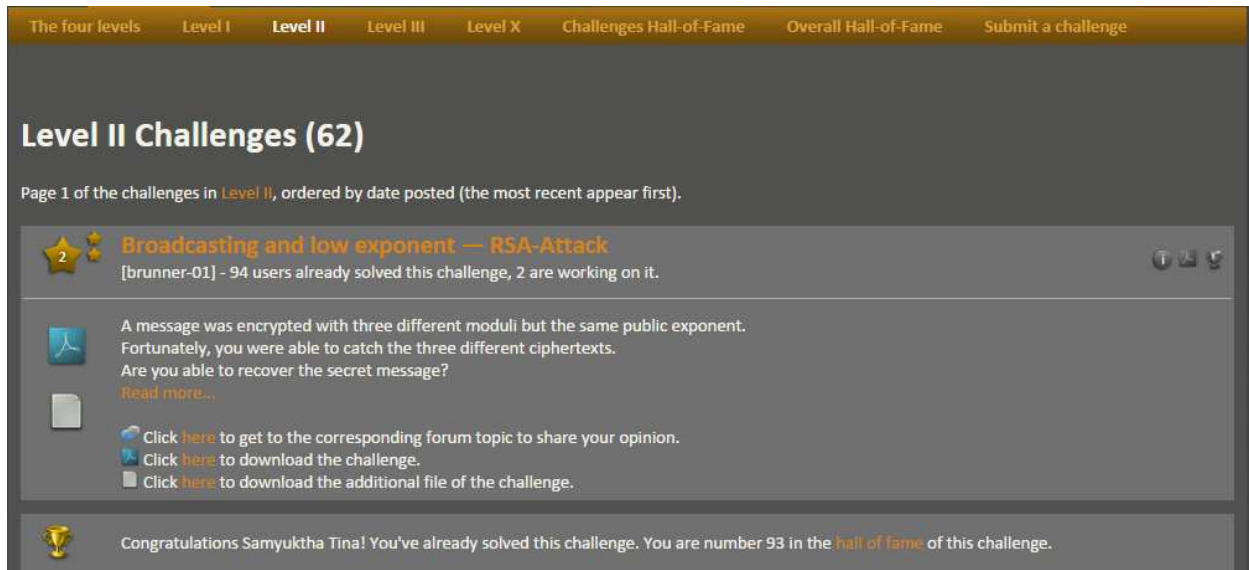


Figure 3: Screenshot success of Broadcast Attack

6. ENHANCEMENTS

This attack on RSA is successful because of the small exponent and the fact that the same message is broadcasted to three different parties.

Samyuktha Sridharan (email.samyu@gmail.com)

Tina Philip (talk2tinaphilip@gmail

Broadcasting And Low Exponent – RSA Attack

The same logic has been applied to solve ‘Alice’s Birthday Party (Part2)’ (Level 2 challenge) . The result shows that the problem has been cracked and the solution was obtained. The below screen shot proves that the solution was obtained.

In the implementation part, similar logic was applied and hex value of the message was obtained. The Hex value is then converted to ASCII and then Base64 decoded to obtain the output as: **Einladung zu meiner Geburtstagsparty. Die Party findet am 20.12.2012 in Bletchley statt.**

From the above message, the location is known to be ‘Bletchley’.

```
C:\Users\Administrator\Desktop\Samyu\CS 265\Project\CRT Enhanced.exe

-----Obtaining PlainText from Broadcast Low Public Exponent Attack-----

The public modulus values
n1=51474516702522238743413237713705671595475072980715144792989428969558728579388909978536904494455862473045694392353612260528582074521711735864082380505874261026769465
877162681931
n2=33245955279991554435602264160544813761707992139183222557892949808060953028449422328281413629912335051440744955455010851012308918294549765005480121061697711447087615
511286414033
n3=655701912126224306905965378166923080547345742776751432326276289177112235232870669540910371386438483343743864812021761599076522036574501373924602203593234785338178963
80654969161
Low public exponent e= 3
The ciphertexts are:
c1=034af0b13ad3b979ee609bc417e0a6a49a424ebd9328a90fe749a8f775edf5880d6c75246c11060d78767b5ef9805c90c3de63e216187e49dd794572de8d339dbf60d9ebf21482a6e7f803696a38fd7a0fb0e
c2=22ed40e8040f2426743f73ed471489283b185e03a2005f017143b0df307e07ac4b63baa4c68da112f695aa961140fdaa97d53b8708975a5fbb154f8b388f14f2f32de34dccc5bde658404707c28833aef69dd
c3=19e02dd25a74adaad2508f3bb0332dd0ca41381d954ade9016d99e33f8aacc0e07df252b4e4650db4ae2668084d4eff2abb460836471571a4f25d9b5737c258db2da13d790999cb483cb2df04e9d74ff7ea59
gcd(n1,n2)=1
gcd(n2,n3)=1
gcd(n3,n1)=1
n1,n2 and n3 are pairwise coprime
N=139c0161af24df1e9d2489b4d112eb62ea3d513391c2e75b2b9183b2f671f86724cf9b1ddeaf1743b7d0c9892bb457043b3ff3149fff8803825ca702c34b8aed091b6808d889aa605a6162e3420fee379fc301b
b4fd77ca9f1b48df685bc756d0daf232c8ca007cf925ea35fdb63d4a68078b6d03ed3e1160e7f2b4fc0041f436e7120185c5f49b67034e137b6fb0bf2431b6a86f5aaa7e0acc8ed16d20275e8b8dc270ba339927
851b26c37c53a2a581a5a7672874c4dccc2a4fcaea1ae8b34drc3f057746082dfbc49b89cc5dae1107ee211ebd386110834e052a55f4263eef3081fbfcc6a86e218f630b3371d49c2b15dd0c3b72a9a965d055
Message = 884cd70bc48d942d3a5a61015364ea351bdf96e500be10036ab3609f57eab2d8551fdhb92ed31fb9a8b542a0a4512e5ed8d67781d8aa73d5f9b710c7fa1513e6d332ffcf95beaf5a83d9e19f163f
bba7a721848e0e03ade22475c397e43b285d460e51eaa4f63eaa57201f0ba6db3aab39ae969b07321fe9dd6f158a08a15e45862a829a498922d6126b9fca076bd4eder2ba1ac0c5e99a590611bad536eab2e
a90cbddc084cfad3c4a1b0551004224650f5da6e3b0874a13615529c23b0d1b1c360hab1d1759e23c43f3d7b362b5e02abbb37204fbd0e328e05b9196ae8ccfb7b0fadb3677fe8
Message Dec in a = 866889435055342491195947185569483141116810800531952410022890723249355551410436069888673370221139237065585627099950744771105820205943661149231276322982
41526328748438724039423209285794206031143809004549939022305276803356358767589556326101320256865181431274984703004599582486297679121042425401628460327599878191387727
2995487229803086492398638520295220473184793816389289894455340931047028564862440974167797075690755124349707435353025035695086847822127415365113507867937266246993078915615
Cuberoor of M , Plain Text in Hex form : 52576c756247466b6457356e494870314947316c6157356c636942485a574a31636e527a6447466e63334268636e52354e694245615755675547467964486b6
Cracked Plaintext = RWlubGPKdV5nIHp1IG1aW5lcIBWJWJcnRcdGFnc3BhcnRSLiBEaUWgUGFpdHk
```

Figure 4: Screenshot of Alice’s Birthday Attack output

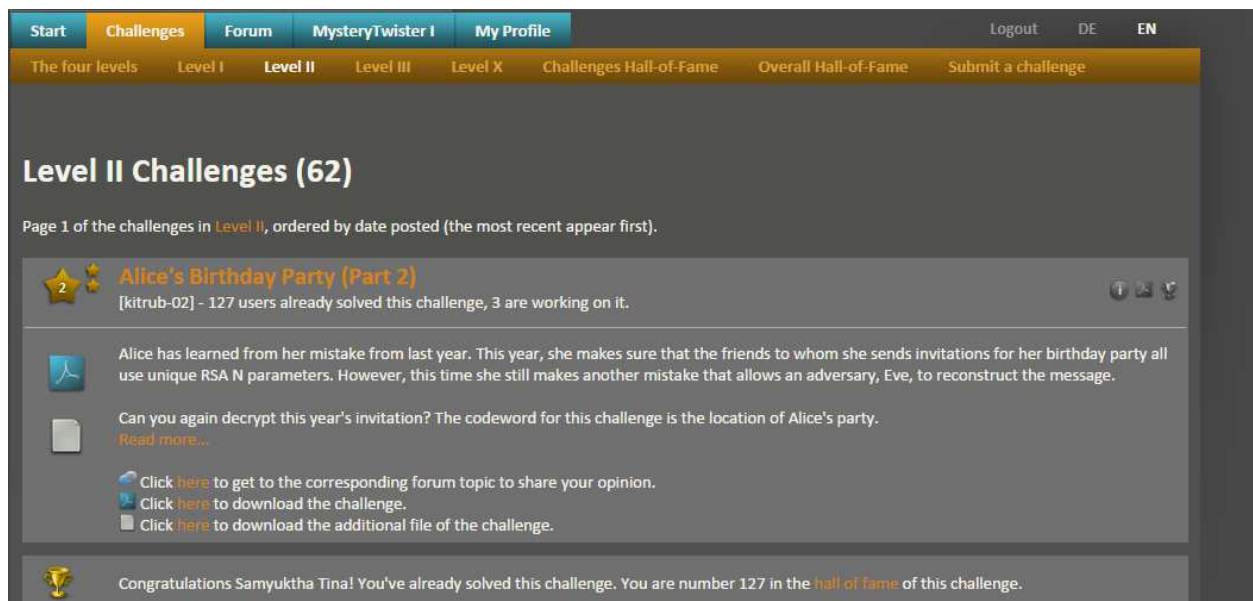


Figure 5: Screenshot success of Alice’s Birthday Attack

Samyuktha Sridharan (email.samyu@gmail.com)

Tina Philip (talk2tinaphilip@gmail

7. SOLUTIONS TO ATTACKS

It is important to know about the possible attacks on RSA algorithm because, RSA algorithm is one of the most widely used public key cryptosystem today. If there exists an attack on RSA system, the security of the application is compromised. Some applications which uses RSA algorithm for security are smartcards, internet applications, etc.

It can be seen that sending in the text without padding can result in trivial attacks. Care should be taken when designing RSA algorithm. RSA algorithm design without padding of random bits should be avoided.

In order to avoid such attack, it is advisable to pad random bits to the public component n , before encryption. Studies have shown that some implementations with padding of timestamp to 'n' also proved to be inefficient. This is because related message attack can be performed for RSA using CRT. The padding has to be done with pseudo-random bits and should be unpredictable.

8. FUTURE WORK

Literature shows that when $e = 3$, for this attack to be successful, it requires that at least 3 same messages to be sent to different parties. It is also evident from previous works that the same message can be sent up to seven people with exponent $e = 3$, in order to retrieve the plain text [2]. It might be possible to try encrypting the same message with $e=3$ and seven different public modulus obtain different cipher text. And then try to retrieve the message using the above implemented attack.

It has been studied that the security of RSA system relies on the difficulty of factorizing large prime numbers and it has been proved that there exist no efficient algorithm to factorize them. It would be reasonable to try to implement a MapReduce paradigm to solve this hard problem. Since it involves large numbers, it could be thought of as splitting the number into different MapReduce phases and obtain the factors [7]. This solution can be useful because MapReduce model provides benefits by providing automatic parallel execution in a distributed environment, fault tolerance, and scales with the input size of the data.

9. CONCLUSION

The implementation successfully attacks the RSA algorithm with the same message and low public exponent. It has been proven that the implementation could be adapted to solve other attacks related to small public exponents. There has been some future work suggested in related to mathematical RSA attacks.

10. REFERENCES

- [1] Rivest, Ronald L., Adi Shamir, and Len Adleman. "A method for obtaining digital signatures and public-key cryptosystems." *Communications of the ACM* 21, no. 2 (1978): 120-126.
- [2] Salah, Imad K., Abdullah Darwish, and Saleh Oqeili. "Mathematical attacks on RSA cryptosystem." *Journal of Computer science* 2, no. 8 (2006): 665.

- [3] Kocher, Paul C. "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems." In *Advances in Cryptology—CRYPTO'96*, pp. 104-113. Springer Berlin Heidelberg, 1996.
- [4] M. WIENER, Cryptanalysis of short RSA secret exponents, *IEEE Trans. Inform. Theory* 36 (1990), 553–558
- [5] Burton, David M. *Elementary number theory*. Tata McGraw-Hill Education, 2006.
- [6] <http://www.di-mgt.com.au/crt.html> , Accessed on October 24, 2014.
- [7] Dean, J. and Ghemawat, S. 2004. MapReduce: Simplified Data Processing on Large Clusters. OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004, 137-150
- [8] Tordable, Javier. "MapReduce for Integer Factorization." *arXiv preprint arXiv:1001.0421* (2010).
- [9] Quisquater, J-J., and Chantal Couvreur. "Fast decipherment algorithm for RSA public-key cryptosystem." *Electronics letters* 18, no. 21 (1982): 905-907.