Project Report

On

**BROADCASTING AND LOW EXPONENT – RSA ATTACK**

*Submitted to*

Prof. Auston Davis

Department of Computer Science

San Jose State University

October 11, 2019

*By:*

**Shamli Singh Edward Broeder**

Graduate Student (SJSU ID: 013962923) Graduate Student (SJSU ID: 007116317)

Contents

[1 Overview 3](#_Toc21625959)

[2 Introduction 3](#_Toc21625960)

[2.1 RSA algorithm 3](#_Toc21625961)

[2.1.1 Components 3](#_Toc21625962)

[2.1.2 Key Generation 3](#_Toc21625963)

[2.1.3 Encryption 3](#_Toc21625964)

[2.1.4 Decryption 3](#_Toc21625965)

[2.2 Example of RSA 4](#_Toc21625966)

[2.3 Cube Root Attack 4](#_Toc21625967)

[2.4 Chinese Remainder Theorem 4](#_Toc21625968)

[2.4.1 Example 5](#_Toc21625969)

[3 Implementation in C 5](#_Toc21625970)

[3.1 Cryptosystem 5](#_Toc21625971)

[3.2 Cryptanalysis 6](#_Toc21625972)

[4 Work Factor 6](#_Toc21625973)

[5 Computation Results 6](#_Toc21625974)

[5.1 Problem on mysterytwisterc3.org 8](#_Toc21625975)

[6 References 9](#_Toc21625976)

# 1 Overview

This project involves decrypting a cipher that was encrypted using RSA algorithm. Instead of the conventional use of a key to decrypt, we will be attacking the cipher instead using cube root attack based on Chinese Remainder Theorem.

# 2 Introduction

## 2.1 RSA algorithm

RSA algorithm is a type of asymmetric cryptosystem, or public key cryptosystem as some like to call. It is named after its inventors Rivest, Shamir and Adleman. It is based on a complex mathematical problem, just like most other cryptosystems.

### 2.1.1 Components

Both sender and receiver have a key pair *(e, d)* where *e* is the encryption component available in public domain, and *d* is the decryption component is secret, known only to the owner.

Another component in the public key is the modulus *N*. Thus, the public key is *(N, e)* while the private key is *(N, d)*.

### 2.1.2 Key Generation

The following outline is used to generate the key for RSA:

* Generate two large, random primes *p* and *q*
* Compute *N = pq* and totient function *ϕ = (p – 1) (q – 1)*
* Choose an integer *e* such that *1 < e < ϕ* and *gcd(e, ϕ) = 1*
* Compute *d* such that *1 < d < ϕ* and *ed = 1 mod ϕ*
* The public key is *(N, e)* and the private key is *d*

### 2.1.3 Encryption

The following outline is used for encryption:

* Get the public key *(N, e)*
* Represent the plaintext as positive integer *M* such that *1 < M < N*
* Compute the ciphertext *C = Me mod N*

### 2.1.4 Decryption

The following outline is used for standard decryption

* Use the private key *(N, d)*
* Compute the message *M = Cd mod N*

## 2.2 Example of RSA

For the sake of simplicity, this example involves simple numbers. Assume we select p = 11 and q = 3 as the two large primes. Then

*N = pq = 11 \* 3 = 33*

And

*ϕ = (p – 1) (q – 1) = 10 \* 2 = 20*

Now we choose encryption exponent *e = 3*, which is relatively prime to *ϕ*.

Also, *ed = 3 \* 7 = 1 mod 20*

Thus, the public key is *(N, e) = (33, 3)* and the private key is *d = 7*.

Now if want to send a message *M = 15*, we compute the ciphertext as

*C = Me mod N = 153 mod 33 = 3375 = 9 mod 33*

To decrypt this ciphertext we use the private key as

*M = Cd mod N = 97 mod 33 = 4782969 mod 33*

*= ((144938 \* 33) + 15) mod 33*

*= 15 mod 33*

## 2.3 Cube Root Attack

It is usual practice to choose a small encryption exponent e to fasten computations. However, such small values like e = 3 can allow easy cube root attack on the RSA encryption. This is especially true if M < N1/3.

In this case, as M < N1/3 mod N has no effect,

C = Me = M3

Thus, M = C1/3

i.e. the cube root of ciphertext will give the plaintext

## 2.4 Chinese Remainder Theorem

Wikipedia states that:

The Chinese remainder theorem is a result about congruencies in number theory, that is if one knows the remainders of the [Euclidean](https://en.wikipedia.org/wiki/Euclidean_division) division of an integer *n* by several integers, then one can determine uniquely the remainder of the division of *n* by the product of these integers, under the condition that the divisors are pairwise coprime.

Mathematically, this theorem can be stated as follows:

Let *n1*, *n2*, *n3*…*nr* be positive integers such that *gcd(ni, nj) = 1* for *i ≠ j*. Then the system of linear congruences is

*x = c1 mod n1*

*x = c2 mod n2*

*…*

*x = cr mod nr*

and has simultaneous solutions unique modulo *n = n1, n2, n3…nr*.

Using Gauss’s algorithm, if *N = n1n2n3…nr*, then

*x = c1N1d1 + c2N2d2 + … + crNrdr (mod N)*

where *Ni = N/ni* and *di = Ni-1 mod ni*

### 2.4.1 Example

Let *n1 = 3, n2 = 4, n3 = 5, c1 = 1, c2 = 2, c3 = 3*

So, *N = 3 \* 4 \* 5 = 60*

*N1 = N/n1 = 20, d1 = 20-1 (mod 3) = 2*

*N2 = N/n2 = 15, d2 = 15-1 (mod 3) = 3*

*N3 = N/n3 = 12, d3 = 12-1 (mod 3) = 3*

Thus, *x = c1N1d1 + c2N2d2 + c3N3d3 (mod N) = ((1 \* 20 \* 2) + (2 \* 15 \* 3) + (3 \* 12 \* 3)) mod 60*

*= (40 + 90 + 108) mod 60 = 238 mod 60 = 58 mod 60*

So x = 58 will satisfy as the solution, such that 0 ≤ x ≤ n1n2n3.

# 3 Implementation in C

We have adopted the BigDigits library in C to implement operations on 512-bit numerical parameters available to us, as traditional C allows 32-bit (int) and 64-bit (long) type operations only.

## 3.1 Cryptosystem

**RSAencryption.c** includes the source code to implement the RSA cryptosystem.

1. Given parameters *n1, n2, n3, e* and plaintext message *M* accepted from user.
2. Convert hexadecimal values of *n1, n2, n3* into decimal using BigDigits library method *bdConvFromHex()*.
3. Check if *(n1, e), (n2, e)* and *(n3, e)* are coprimes.
4. Encrypt the message M into 3 different ciphertexts using BigDigits library method *bdModExp\_ct()*.
5. Print the ciphertexts *c1, c2, c3* in hexadecimal format using BigDigits library method *bdPrintHex()*.

To execute this program, use the command:

$ gcc RSAencryption.c lib/bigd.c lib/bigd.h lib/bigdigits.c lib/bigdigits.h lib/bigdtypes.h -o encrypt

$ ./encrypt

## 3.2 Cryptanalysis

**crt.c** includes the source code to implement RSA cryptanalysis using Chinese Remainder Theorem.

1. Given three ciphertexts, and three known moduli. e is known to be 3.

2. Convert hex values of plaintext and moduli to using BigDigits library method bdConvFromHex().

3. Multiply the three moduli together using bdMultiply, to get the product. Then multiply each modulus against each other modulus, to get product1, product2, and product3.

4. Using bdModInv, get the inverse modulo of each given modulus and corresponding product.

5. Using bdModMult, compute product of each ciphertext and each product mod the product of the three given moduli. Then use bdAdd to add these up.

6. Then use bdCubeRoot to find the cube root of that result.

7. Convert this back to hex, then print it as ascii text. This prints the plaintext, decrypted from the given ciphertext, using the Chinese Remainder Theorem.

To execute this program, use the command:

$ gcc crt.c lib/bigd.c lib/bigd.h lib/bigdigits.c lib/bigdigits.h lib/bigdtypes.h -o decrypt

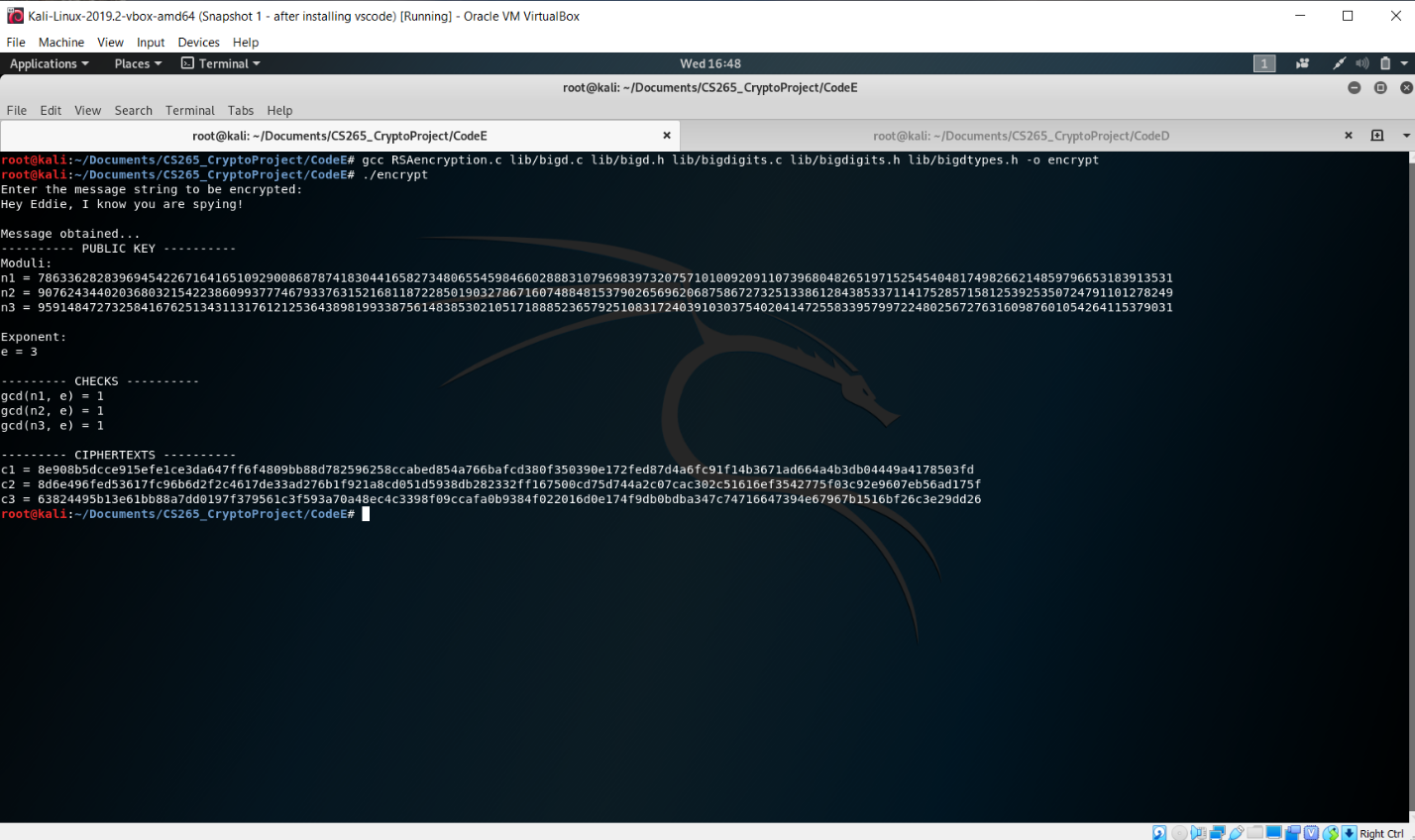
$ ./decrypt

# 4 Work Factor

The runtime complexity of crt.c is of the order of n, where n is the size of the ciphertext. The memory usage is minimal as well, using a constant number of about 15 BIGD type variables.

# 5 Computation Results

On executing cryptosystem program, we get three ciphertexts for the message entered by the user.



The moduli are given to us as:

n1 = 009623511e6769644d693e89f692ffc2558eef121d42ca98699781e139e29c2e1aa58d8883bbdba41165fdeb85a9a5648fc29a65d59e9401694dd11ae205f0ce3b

n2 = 00ad4bc0f980f4523f490fc40c12efcecc1e8af67890b6562449876e8e091e861cda699e5a8eb309b0a9d6b293100c1229fbd18a5951f33b6fbab1fd8d90f7c829

n3 = 00b7223364d88353ec02b0850e8a01d2ba9ca2663c32c15df7b596406c6fc1c171ac965a554b8b338f4bb046c543937b4b19c699864f1d0dd4be0177eccce0bb57

Our encryption exponent is *e = 3*.

We entered *M =* ***Hey Eddie, I know you are spying!***

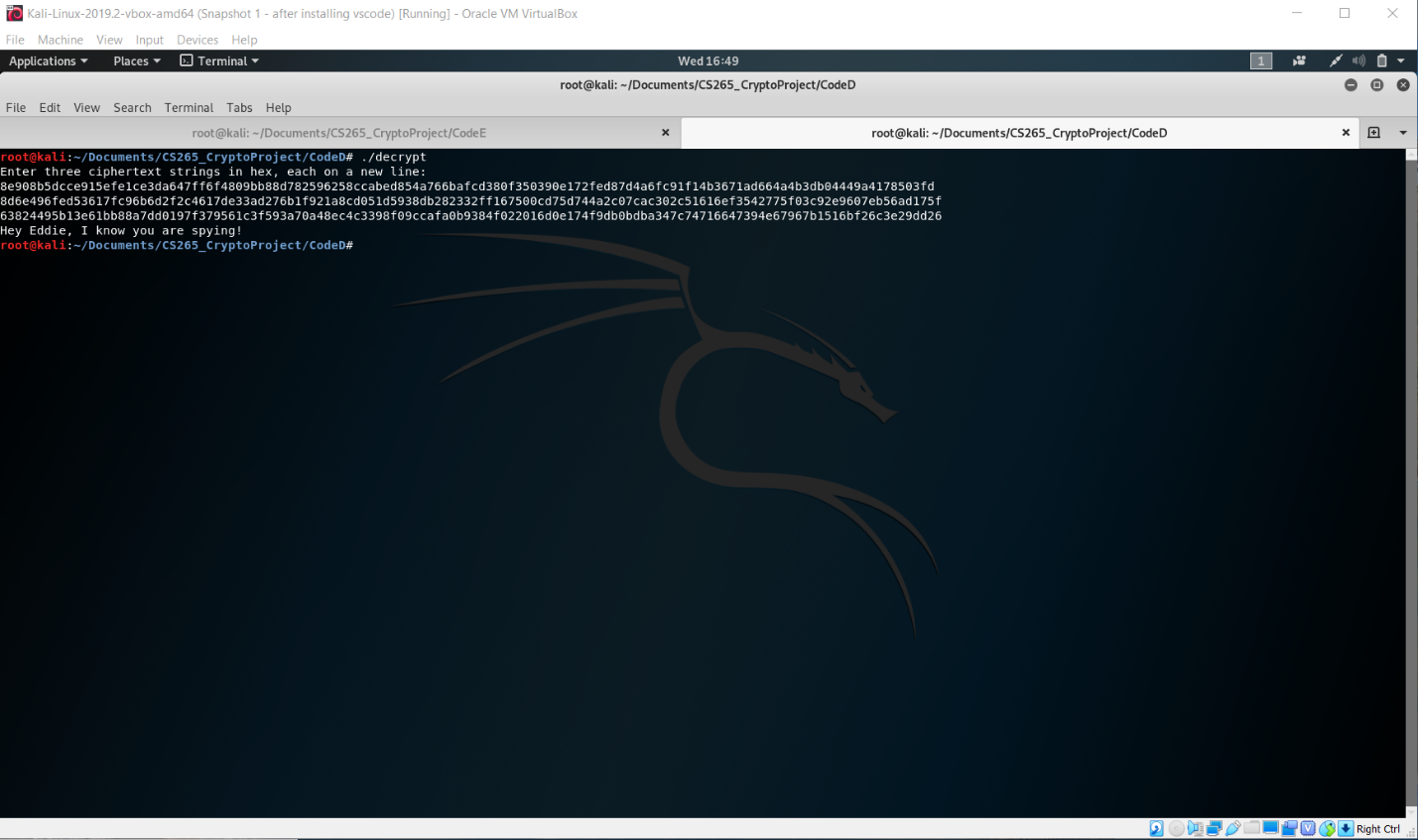
We got the three ciphertexts as:

c1 = 8e908b5dcce915efe1ce3da647ff6f4809bb88d782596258ccabed854a766bafcd380f350390e172fed87d4a6fc91f14b3671ad664a4b3db04449a4178503fd

c2 = 8d6e496fed53617fc96b6d2f2c4617de33ad276b1f921a8cd051d5938db282332ff167500cd75d744a2c07cac302c51616ef3542775f03c92e9607eb56ad175f

c3 = 63824495b13e61bb88a7dd0197f379561c3f593a70a48ec4c3398f09ccafa0b9384f022016d0e174f9db0bdba347c74716647394e67967b1516bf26c3e29dd26

We now execute the cryptanalysis program with input c1, c2 and c3 from earlier step



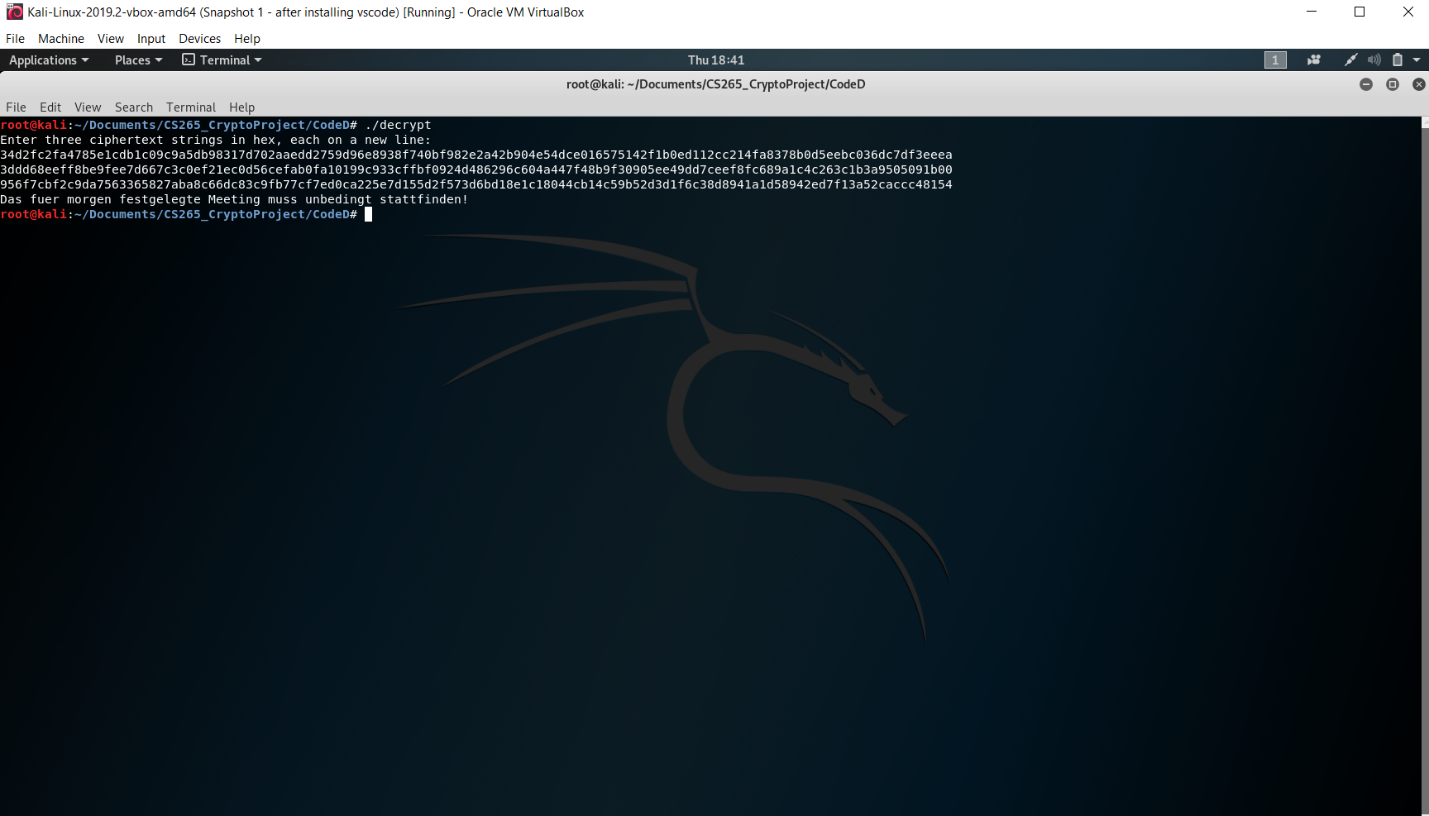
## 5.1 Problem on mysterytwisterc3.org

Based on the system that we have developed, we solved for the ciphertext problem given on mysterytwisterc3.org for the ciphertexts:

c1 = 34d2fc2fa4785e1cdb1c09c9a5db98317d702aaedd2759d96e8938f740bf982e2a42b904e54dce016575142f1b0ed112cc214fa8378b0d5eebc036dc7df3eeea

c2 = 3ddd68eeff8be9fee7d667c3c0ef21ec0d56cefab0fa10199c933cffbf0924d486296c604a447f48b9f30905ee49dd7ceef8fc689a1c4c263c1b3a9505091b00

c3 = 956f7cbf2c9da7563365827aba8c66dc83c9fb77cf7ed0ca225e7d155d2f573d6bd18e1c18044cb14c59b52d3d1f6c38d8941a1d58942ed7f13a52caccc48154



We got the message M = ***Das feur morgen festgelegte Meeting muss unbedingt stattfinden!***

# 6 References

[1] ***“Information Security: Principles and Practice”***, M. Stamp, Wiley-IEEE Press, August 2019.

[2] “***The Handbook of Applied Cryptography”***, by A. Menezes, P. van Oorschot, and S. Vanstone, CRC Press, 1996.

[3] <https://en.wikipedia.org/wiki/Chinese_remainder_theorem>

[4] <https://www.di-mgt.com.au/crt.html>

[5] <https://www.di-mgt.com.au/bigdigits.html>