Optimizing Security in Resource-Constrained Systems

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering with specialization in Information Security

by

SANKALP MEHANI **16BCI0188**

ANANT SINGHAL **16BCI0159**

Under the guidance of
Prof. Sivanesan S
SCOPE
VIT, Vellore

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

May, 2020

DECLARATION

I hereby declare that the thesis entitled “Optimizing Security in Resource-Constrained Systems" submitted by us, for the award of the degree of Bachelor of Technology in Computer Science and Engineering with specialization in Information Security to VIT is a record of bonafide work carried out by me under the supervision of Mr. Sivanesan S.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date :

Signature of the Candidate

CERTIFICATE

This is to certify that the thesis entitled "Optimizing Security in Resource-Constrained Systems" submitted by **Sankalp Mehani (16BCI0188) and Anant Singhal (16BCI0159)**, **SCOPE**, VIT, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering with specialization in Information Security*, is a record of bonafide work carried out by them under my supervision during the period, 01. 12. 2019 to 29.05.2020, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date :

Signature of the Guide

Internal Examiner

External Examiner

Head of the Department

CSE with specialization in Information Security

ACKNOWLEDGEMENTS

We take this opportunity to thank the almighty God, the author of knowledge and wisdom for gracing upon us for the successful accomplishment of our Capstone Project.

We express utmost gratefulness to our chancellor, Dr G. Viswanathan and every member of the management staff of VIT University, for providing us with a conducive study environment.

We also wish to express our extreme depth of gratitude to the Dean of Computer Science and Engineering, Prof. Santhi V, for providing us with an opportunity to carry out this project work successfully.

We would like to thank the Computer Science and Engineering with specialization in Information Security Head of Department, Prof. Prabu S, and all the faculty members of VIT who encouraged and helped us during the course of the project.

We would like to express our gratitude to our project guide, Prof. Sivanesan S, for assisting us both technically and morally, giving valuable suggestions and continuously encouraging us throughout the course of this project.

We would like to thank our project coordinator, Prof. Ramani S for his encourage and support.

We would like to thank each and every member of the teaching and non-teaching staff for their valuable help in all aspects.

Finally, yet importantly, we would also like to thank our family and friends for their unconditional support and inspiration since the inception of the project.

Sankalp Mehani and Anant Singhal

Executive Summary

Ron Rivest, Adi Shamir, and Leonard Adleman of Massachusetts Institute of Technology made history in 1977 when they, after trying numerous approaches over the course of a year, published the paper which introduced RSA to the world. It would be an understatement to call the paper revolutionary as they pioneered the field of asymmetric public-private key cryptosystems. RSA is now one of the most widely used algorithms for secure data transmission. While Rivest and Shamir were computer scientists, Leonard Adleman was a mathematician and evidently, there is a lot of mathematics involved in the working of RSA. It is very thought provoking that with numerous advancements in the field of mathematics, could there be a way to still improve the quintessential key-exchanging algorithm. This project aims to not only ponder this question but also to answer it with a unique implementation. It also makes one realize the fact that however revolutionary a thing may be, there is always a room for improvement that pushes it further towards the ultimate destination of absolute perfection.

Contents

ACKNOWLEDGEMENTS ii

Executive Summary iii

List of Figures v

List of Tables vi

List of Abbreviations vii

Symbols and Notations viii

1. INTRODUCTION 1

1.1 AIM 1

1.2 OBJECTIVE 1

1.3 MOTIVATION 1

1.4 BACKGROUND 2

2. PROJECT DESCRIPTION AND GOALS 5

3. TECHNICAL SPECIFICATION 7

4. DESIGN APPROACH AND DETAILS 11

4.1 DESIGN APPROACH 11

4.2 CONSTRAINTS, ALTERNATIVES AND TRADEOFFS 17

5. SCHEDULE, TASKS AND MILESTONES 19

6. PROJECT DEMONSTRATION 22

7. RESULTS AND DISCUSSION 28

8. SUMMARY 31

9. REFERENCES 32

APPENDIX A 33

List of Figures

Figure 1: Landing Page of VirusShare.com 8

Figure 2: List of pages containing Virus Signatures 8

Figure 3: Some signatures from a page 9

Figure 4: Karatsuba Flowchart....................13

Figure 5: Fast Exponentiation Flowchart 15

Figure 6: VIT Antivirus working flowchart 17

Figure 7: Schedling of the project....................19

Figure 8: Gantt Chart of the project....................21

Figure 9: Project Flowchart 22

Figure 10: MS Visual Studio Workbench 23

Figure 11: Selecting a file for scanning....................23

Figure 12: File signature not found in dictionary....................24

Figure 13: File becomes clean 24

Figure 14: PC implementation of Optimized RSA 1 25

Figure 15: PC implementation of Optimized RSA 2 25

Figure 16: Smartphone implementation of Optimized RSA 1....................26

Figure 17: Smartphone implementation of Optimized RSA 2....................27

Figure 18: PC implementation of normal RSA 1 28

Figure 19: PC implementation of normal RSA 2 28

Figure 20: Smartphone implementation of normal RSA 1 29

Figure 21: Smartphone implementation of normal RSA 2 30

List of Tables

Table 1: Comparison of different RSA implementations on different devices 30

List of Abbreviations

RSA	Algorithm devised by Rivest, Shamir and Adleman
MD5	Latest version of the Message Digest Algorithm
RAM	Random Access Memory
IOT	Internet of Things
SHA	Secure Hash Algorithm
CPS	Cyber Physical Systems

Symbols and Notations

s	seconds
ms	milliseconds
&	And

1. INTRODUCTION

1.1 AIM

The fundamental aim of this project is to analyze two unrelated mathematical techniques and use them correspondingly in different parts of the RSA algorithm to accelerate and optimize it. Furthermore, an antivirus software is aimed to be constructed from scratch that will provide basic features like scanning and retrieving signatures from files. Finally, a comparative conclusion needs to be illustrated in order to quantify the achieved efficiency.

1.2 OBJECTIVE

The primary objective of this project is to propose a system that optimizes the security and increases the efficiency of a network of resource-constrained devices by implementing cost cutting techniques to the algorithm used and creating an application to facilitate the work. The objectives are further elaborated below:

- Define and Analyze the RSA Algorithm
- Optimize the RSA algorithm by introducing the Karatsuba and Fast Exponentiation multiplicative methods to reduce time complexity.
- Create an antivirus program to scan a file and retrieve the hash codes from the said file.
- Use the retrieved hash code as an input for the optimized RSA algorithm and implement the algorithm.
- Draw a comparison between the traditional RSA algorithm and the optimized one by comparing run times.

1.3 MOTIVATION

The main source of motivation for this project was the fact that RSA, being one of the most used security algorithm, still has a room for improvement and the knowledge of some techniques that could be used for the improvement. Also, to reduce the complexity, we needed a basic antivirus and the market is filled with all these complicated, feature-filled antivirus programs and this motivated us to create a simple antivirus of our own. Also, a complete implementation of proposed work could be used in various scenarios like key exchanging, IOT device authentication etc.

1.4 BACKGROUND

In today’s world, RSA is that the market leader for cybersecurity, secure data transmission and digital risk management solutions. it is a quite an asymmetric cryptographic algorithm and is widely employed in key exchanges. Since one all told the two keys employed in RSA is public, meaning it'll incline to anyone for encryption and also the opposite one is private, meaning only party has the access thereto and only they'll use it for decryption, it falls under the category of public key cryptography.

Definition of Modular multiplication can be stated as the calculation of $P = A \times B$ (mod m), where A, B and m are integers and $0 \leq A, B < m$ where A,B are two positive integers. Multiplying of squaring operation on a number is considered to be solely an easy multiplication of two numbers. Several techniques can be used to obtain multiplication results like Brickell’s method, multiply then divide, interleaving multiplication and reduction, Completion of modular multiplication is achieved by repeatedly subtracting modulus from the multiplicand until and unless we get the result as smaller that the given modulus. As the size of the modulus increases, the stated method starts consuming a lot of time. Dividing the modulus is another way of obtaining modular multiplication but, again it also consumes a lot of time.. Modular multiplication is an important function while performing Modular exponentiation (a^n mod m). Since modular multiplications are one of the most time-consuming process and operations for generating ciphertext and getting back their plaintext, optimizing them becomes important. Optimization of this can be done if we reduce the latency of each modular multiplication involved or by simply reducing the modular multiplications we are using.

J. Vuillemin and M. Shand in “Fast Implementation of RSA Cryptography” detail and analyze the critical techniques which might be combined within the structure of fast hardware for RSA cryptography: Chinese remainders, asynchronous carry completion adders, star chains, Hensel's odd division (a.k.a. Montgomery modular multiplication reduction), quotient pipe-lining, carry-save representation. A gain within the performance software with every enhancement of the performance of hardware of RSA which was also analyzed by them. Additionally, to the techniques enumerated above, their best software implementation of RSA involved Karatsuba multiplication and specific squaring. This was the fastest commercially available RSA in 1992 due to the strategies implemented.

Most antivirus software is big and complex in nature. Different antivirus software has different approaches to how they scan, detect, quarantine, and disinfect an infected file. The approach is usually like this:
Scan an application, file, or program and compute some kind of code through it. That code is then compared with a humongous set of codes stored in its database. If a code that's identical or just like the one that has been retrieved is found, the file is claimed as infectious. The corresponding code is taken into consideration to be that of malware and also the file is then quarantined to review or simply removed if deemed impossible to wash.

When we click on an .EXE file, it doesn't just open immediately, although it should seem to be it on a faster computer. The antivirus software installed on the pc does a preliminary check to create sure the file is safe before letting it open. Modern antivirus software also implements "heuristic" checking to analyze any kind of bad behavior that the file is reflecting which might end in a replacement, unknown virus.

Although every antivirus within the globe follows the identical fundamental steps described above, they differ on factors a bit like the formation of the database, pre, and post scanning procedures, quarantine and removal norms, etc. But one thing that has not been done yet is that the combination of
(a) the foremost fundamental feature of an antivirus program i.e. the scanning and retrieval of hash signatures from a given file and,
(b) the massively popular RSA algorithm.

Review of literature suggests that a number of studies have been carried out to study various approaches to optimize the RSA algorithm. As evident from the studies, there have been multiple possible techniques that could be applied to achieve optimization in different sectors. With the increasing usage of modern handheld and other IOT devices (most of which are resource constrained), there is a need to do a comprehensive and thorough study on the said algorithms and find out the one most suitable for small devices using practical comparative factors.

2. PROJECT DESCRIPTION AND GOALS

The project can be described in the following two modules:

Optimized RSA module: RSA is one of the best key exchange and encryption algorithms. It is one of the first used public-key cryptosystems. In this algorithm, the encryption key is public and the decryption key is kept private and hence it is an asymmetric cryptosystem. It is justifiably very productive and popular in the security community, however, like many asymmetric cryptosystems, it uses a lot of computation power of the system as it deals with very large prime numbers and executes operations on them. In this project, we shall be discussing and showcasing two unrelated methods for accelerating different parts of the RSA algorithm namely, 'Karatsuba Method for Fast Multiplication' and 'Fast Exponentiation or Squaring Method'

VIT Antivirus: Our very own antivirus has been efficiently narrowed down to the basics of any antivirus system. It does what almost every major antivirus software does but without the redundant functionality, which makes it the perfect suit for our project which prefers optimization and efficiency over fancy features.

The goal of this project is to build an optimized security mechanism for applications and devices that are equipped with relatively less quantifiable resources like battery, storage capacity, processing power etc. often known as resource-constrained devices. Some examples of resource-constrained systems are Smart Watches, Mobile Cyber Physical Systems (CPS), Small scale smartphones and Smart Grid Devices. The proposed mechanism should provide improved security making use of the limited resources. Multiple testing iterations of the project need to be done in order to find and fix bugs and logical errors that may happen during the design and development phase. The goals with which the project is designed are as follows:

1. Computation Complexity: Since many resource-constrained systems are not richly equipped with enough RAM or powerful processors, the computation complexity of the system can be very high if the security algorithm has

many rounds or has a very long key. So, the security algorithm should be optimized accordingly.

2. Data manipulation speed: Key generation and manipulation of data for encryption or decryption should be quicker.

3. Cost Feasibility: The cost of the implemented mechanism should be low as should be the requirements for running the software so as to facilitate the production of software at the minimum cost possible and thereby, making it possible for the clients to purchase the software for their infrastructure with high optimization and low cost, both monetary and resource-wise.

4. Space Consumption: Since the algorithm is modified to be implemented on resource constrained systems having less storage capacity, the space complexity of the algorithm should be less.

3. TECHNICAL SPECIFICATION

The VIT Antivirus will be constructed using the .NET Framework in Microsoft Visual Studio 2015 RC. The following namespaces will be imported:

System.IO: Allows us to read, write and manipulate files and data streams.

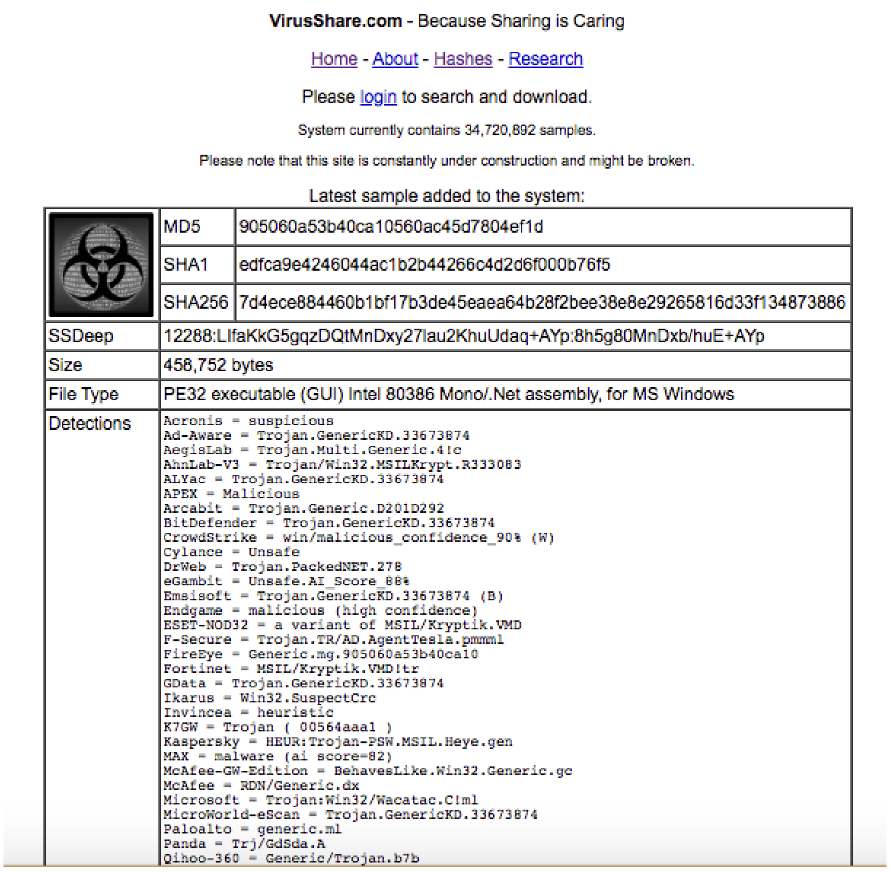
System.Security: provides the underlying structure of the common language runtime security system, it also provides base classes for permissions.

System.Security.Cryptography: provides services like generation of ciphertext of a data and back to its plaintext. It also provides operations like message authentication, random number generation and hashing.

Also, the following functions/methods will be implemented:

md5_hash ( ):	
hash_generator ( ):	The function will obtain the desired hash of the file that we put into our antivirus.
OpenRead ( ):	The function opens the file that matches with the name that is passed while calling the function.
ComputeHash ( ):	Computes the hash of the filestream that has been passed as argument.
PrintByteArray ( ):	Will get the array of bytes and it will be converted to hexadecimal format before it can be read easily.

To get all known virus signatures, the database from www.virusshare.com will be used which contains close to 20 million virus signatures.



VirusShare.com - Because Sharing is Caring

Home - About - Hashes - Research

Please login to search and download.

System currently contains 34,720,892 samples.

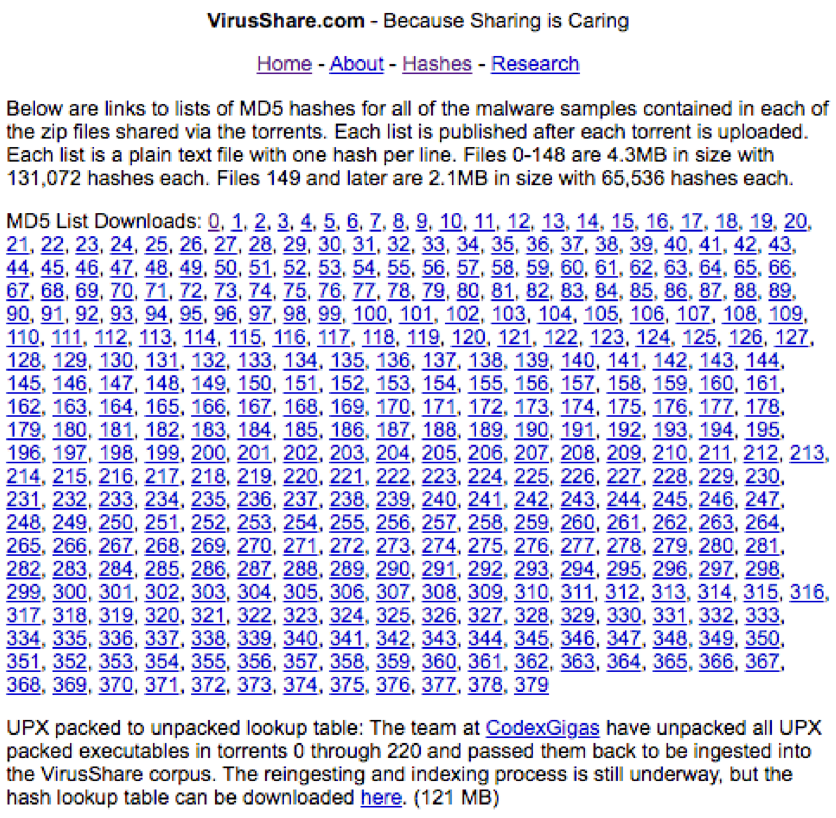
Please note that this site is constantly under construction and might be broken.

Latest sample added to the system:

	MD5	905060a53b40ca10560ac45d7804ef1d
	SHA1	edfca9e4246044ac1b2b44266c4d2d6f000b76f5
	SHA256	7d4ece884460b1bf17b3de45eaea64b28f2bee38e8e29265816d33f134873886
SSDeep	12288:LIfaKkG5gqzDQtMnDxy27lau2KhuUdaq+AYp:8h5g80MnDxb/huE+AYp	
Size	458,752 bytes	
File Type	PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows	

Detections	Acronis = suspicious
	Ad-Aware = Trojan.GenericKD.33673874
	AegisLab = Trojan.Multi.Generic.4!c
	AhnLab-V3 = Trojan/Win32.MSILKrypt.R333083
	ALYac = Trojan.GenericKD.33673874
	APEX = Malicious
	Arcabit = Trojan.Generic.D201D292
	BitDefender = Trojan.GenericKD.33673874
	CrowdStrike = win/malicious_confidence_90% (W)
	Cylance = Unsafe
	DrWeb = Trojan.PackedNET.278
	eGambit = Unsafe.AI_Score_88%
	Emsisoft = Trojan.GenericKD.33673874 (B)
	Endgame = malicious (high confidence)
	ESET-NOD32 = a variant of MSIL/Kryptik.VMD
	F-Secure = Trojan.TR/AD.AgentTesla.pmmml
	FireEye = Generic.mg.905060a53b40ca10
	Fortinet = MSIL/Kryptik.VMD!tr
	GData = Trojan.GenericKD.33673874
	Ikarus = Win32.SuspectCrc
	Invincea = heuristic
	K7GW = Trojan (00564aaa1)
	Kaspersky = HEUR:Trojan-PSW.MSIL.Heye.gen
	MAX = malware (ai score=82)
	McAfee-GW-Edition = BehavesLike.Win32.Generic.gc
	McAfee = RDN/Generic.dx
	Microsoft = Trojan:Win32/Wacatac.C!ml
	MicroWorld-eScan = Trojan.GenericKD.33673874
	Paloalto = generic.ml
	Panda = Trj/GdSda.A
	Qihoo-360 = Generic/Trojan.b7b	

Figure 1: Landing Page of VirusShare.com



VirusShare.com - Because Sharing is Caring

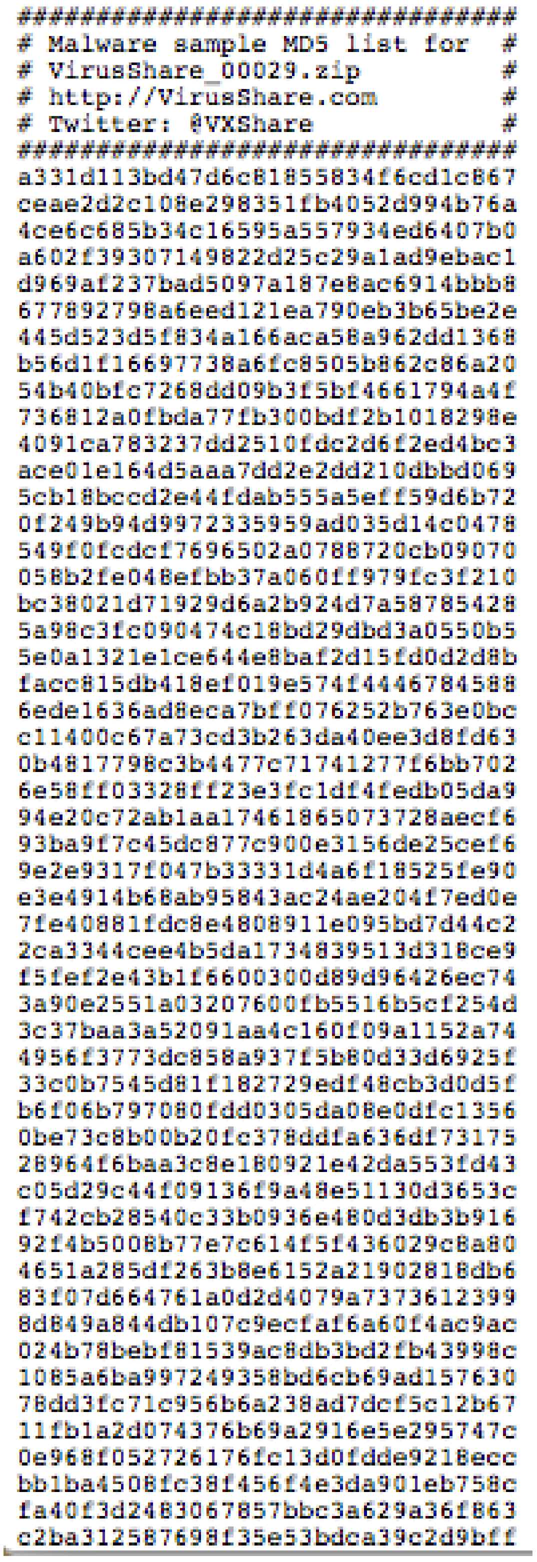
Home - About - Hashes - Research

Below are links to lists of MD5 hashes for all of the malware samples contained in each of the zip files shared via the torrents. Each list is published after each torrent is uploaded. Each list is a plain text file with one hash per line. Files 0-148 are 4.3MB in size with 131,072 hashes each. Files 149 and later are 2.1MB in size with 65,536 hashes each.

MD5 List Downloads: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379

UPX packed to unpacked lookup table: The team at CodexGigas have unpacked all UPX packed executables in torrents 0 through 220 and passed them back to be ingested into the VirusShare corpus. The reingesting and indexing process is still underway, but the hash lookup table can be downloaded here. (121 MB)

Figure 2: List of pages containing Virus Signatures



```
###################################
# Malware sample MD5 list for    #
# VirusShare_00029.zip           #
# http://VirusShare.com          #
# Twitter: @VXShare              #
###################################
a331d113bd47d6c81855834f6cd1c867
ceae2d2c108e298351fb4052d994b76a
4ce6c685b34c16595a557934ed6407b0
a602f39307149822d25c29a1ad9ebac1
d969af237bad5097a187e8ac6914bbb8
677892798a6eed121ea790eb3b65be2e
445d523d5f834a166aca58a962dd1368
b56d1f16697738a6fc8505b862c86a20
54b40bfc7268dd09b3f5bf4661794a4f
736812a0fbda77fb300bdf2b1018298e
4091ca783237dd2510fdc2d6f2ed4bc3
ace01e164d5aaa7dd2e2dd210dbbd069
5cb18bccd2e44fdab555a5eff59d6b72
0f249b94d9972335959ad035d14c0478
549f0fcdcf7696502a0788720cb09070
058b2fe048efbb37a060ff979fc3f210
bc38021d71929d6a2b924d7a58785428
5a98c3fc090474c18bd29dbd3a0550b5
5e0a1321e1ce644e8baf2d15fd0d2d8b
facc815db418ef019e574f4446784588
6ede1636ad8eca7bff076252b763e0bc
c11400c67a73cd3b263da40ee3d8fd63
0b4817798c3b4477c71741277f6bb702
6e58ff03328ff23e3fc1df4fedb05da9
94e20c72ab1aa17461865073728aecf6
93ba9f7c45dc877c900e3156de25cef6
9e2e9317f047b33331d4a6f18525fe90
e3e4914b68ab95843ac24ae204f7ed0e
7fe40881fdc8e4808911e095bd7d44c2
2ca3344cee4b5da1734839513d318ce9
f5fef2e43b1f6600300d89d96426ec74
3a90e2551a03207600fb5516b5cf254d
3c37baa3a52091aa4c160f09a1152a74
4956f3773dc858a937f5b80d33d6925f
33c0b7545d81f182729edf48cb3d0d5f
b6f06b797080fdd0305da08e0dfc1356
0be73c8b00b20fc378ddfa636df73175
28964f6baa3c8e180921e42da553fd43
c05d29c44f09136f9a48e51130d3653c
f742cb28540c33b0936e480d3db3b916
92f4b5008b77e7c614f5f436029c8a80
4651a285df263b8e6152a21902818db6
83f07d664761a0d2d4079a7373612399
8d849a844db107c9ecfaf6a60f4ac9ac
024b78bebf81539ac8db3bd2fb43998c
1085a6ba997249358bd6cb69ad157630
78dd3fc71c956b6a238ad7dcf5c12b67
11fb1a2d074376b69a2916e5e295747c
0e968f052726176fc13d0fdde9218ecc
bb1ba4508fc38f456f4e3da901eb758c
fa40f3d2483067857bbc3a629a36f863
c2ba312587698f35e53bdca39c2d9bff
```

Figure 3: Some signatures from a page

For the optimization and implementation of the revised RSA algorithm, the systems used had the following specifications:

Laptop Configurations:

CPU: Intel Core i5 processor

Clock Speed: 2.5 GHz

RAM: 8 GB

Operating System: Windows 10 (64-bit)

Compiler: GCC Compiler

IDE: CodeBlocks

Mobile Phone Configurations:

Phone Used: Micromax Bolt

CPU: Mediatek processor

Clock Speed: 1.2 GHz

RAM: 512 MB

Operating System: Android 4.4.2 (Kitkat)

Compiler: GCC Compiler (offline)

App Used: Coding C++

4. DESIGN APPROACH AND DETAILS

4.1 DESIGN APPROACH

1. Karatsuba Method for Fast Multiplication

Suppose we have two given binary strings which represent the values of the two integers. Now, our objective is finding the multiplication of these two binary strings. Let us take an example, let the first binary string be “1111” & “1001” be the other binary string then, the obtained output is 135.
For better explanation and simplicity, we assume the length of the two binary strings to be the same as n.

One of the Naïve Approaches would be to follow the process and steps we studied in school i.e. multiplying two numbers by taking all the bits from the second number one by one & multiplying it with the bits in the first number and then, adding each resulted multiplication as we learned in school. However, this algorithm is heavy on the time constraint and takes O(n^2) time.

Working of this approach:

```
  X=101001=41
  Y=101010=42
  ___________
      1010010
      101001
    +101001
  ___________
11010111010=1722
```

Multiplication of the two binary integers consuming less time can be obtained if we instead use the **Divide and Conquer** approach. Let us take the numbers to be multiplied as X and Y. Now, these numbers are supposed to be divided into two equal halves. Let us assume that the length (number of digits) **n** of the number is even for our simplicity.

X = Xl*2^(n/2) + Xr [Xl and Xr contain leftmost and rightmost n/2 bits of X]

Y = Yl*2^(n/2) + Yr [Yl and Yr contain leftmost and rightmost n/2 bits of Y]

Then, the product XY can be re-written as follows:

XY = (Xl*2^(n/2) + Xr) (Yl*2(n/2) + Yr)
= 2n XlYl + 2^(n/2) (XlYr + XrYl) + XrYr

Now, if we look at this formula, then we can see that there are four multiplications of the size of (n/2). Hence, the problem of size n is basically divided into four sub-problems of size (n/2) but that doesn’t generally mean that it will help since the solution of recurrence $T(n) = 4T(n/2) + O(n)$ is $O(n^2)$.

So, the most important part for this algorithm shall be discussed. The terms in the middle should be changed in such a way that only one extra product is enough. Then, the important expression for the middle two terms will be:

XlYr + XrYl = (Xl + Xr) (Yl + Yr) - XlYl- XrYr

There, the final value of XY becomes:

XY = 2n XlYl + 2^(n/2) * [(Xl + Xr) (Yl + Yr) - XlYl - XrYr] + XrYr

So, with this trick, the recurrence changes to

T(n) = 3T(n/2) + O(n)

and solution of recurrence becomes **O (n1.59).**

Now, let us consider a different case where the length of the binary strings aren’t equal and they aren’t even as well. In this case, we append **0**s at the beginning to make the length equal. Now, for handling odd lengths, floor(n/2) bits are put in the left half of the string and n/2 bits are put in the right half of the binary string. Therefore, the expression becomes:

XY = 22^(n/2) XlYl + 2^(n/2) * [(Xl + Xr) (Yl + Yr) - XlYl - XrYr] + XrYr

In our project, Karatsuba multiplication is used for calculating n and φ where,

n = P*Q [P and Q are two large prime numbers]

φ = (P-1) *(Q-1)

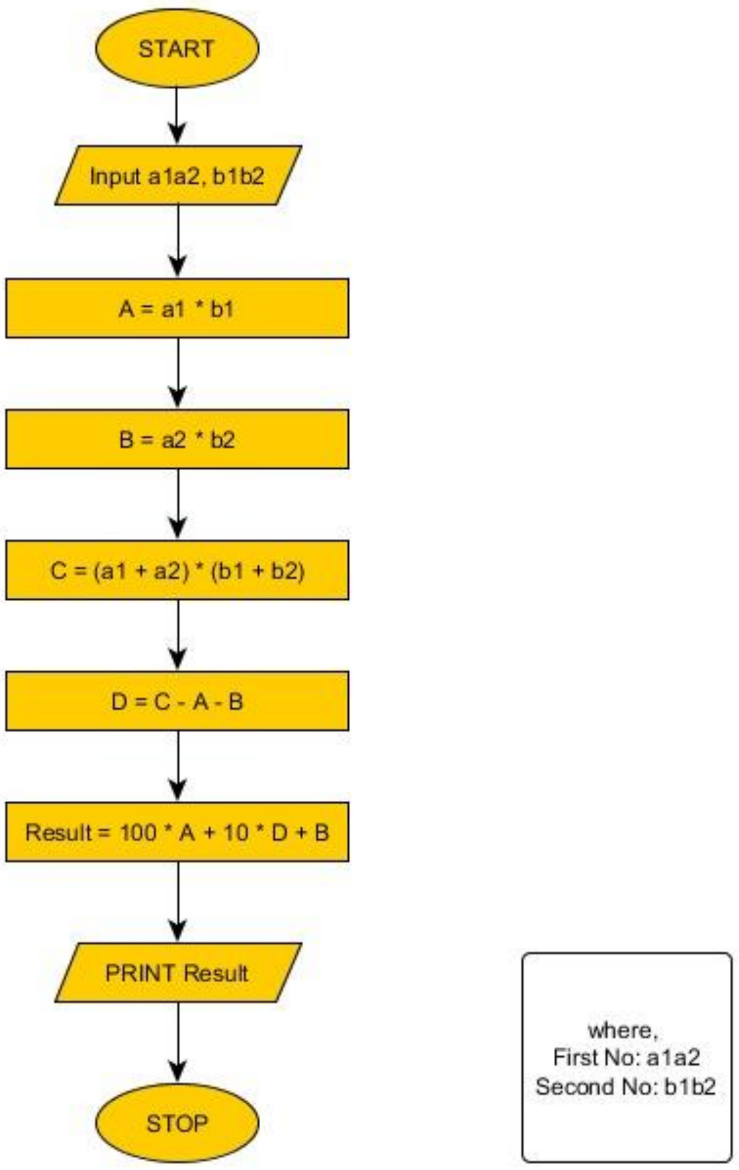


Figure 4: Karatsuba Flowchart

2. Fast Exponentiation Method

Exponentiation by squaring is a frequently used method for computing large integer powers of a number in computer programming and mathematics. This method can be used in powering of matrices or in modular arithmetic.
Fast Exponentiation method is as follows:

If a and b are even then, for any a^b, we can write it as $(a^{b/2})^2$. On the other hand, if b is odd, we can write the expression as a* $(a^{(b-1)/2})^2$.

$$x^n = \begin{cases} 1, & \text{if } n = 0 \\ \frac{1}{x}^{-n}, & \text{if } n < 0 \\ x \cdot \left(x^{\frac{n-1}{2}}\right)^2, & \text{if } n \text{ is odd} \\ \left(x^{\frac{n}{2}}\right)^2, & \text{if } n \text{ is even} \end{cases}$$

We can see that we are cutting the problem into half at each step, adding an extra multiplication for odd numbers.
The algorithm looks like this:

```
int exp (int x, int y) {

     if (y == 1)
         return x;
     else if (y == 2)
         return x*x;

     else if (y%2==0)
     {
         return exp (exp (x, y/2),2);
     }
     else
     {
         return x*exp (exp (x, (y-1)/2),2);
     }
}
```

A brief analysis of this formula allows us to visualize that this method uses only O(log2n) squarings and O(log2n) multiplications!

In our project, Fast Exponentiation Method is used for calculating M^e and C^D where M is the Plain Text, e is the part of the encryption key (public key), C is the obtained Cipher text and D is the part of the Decryption key (private key).

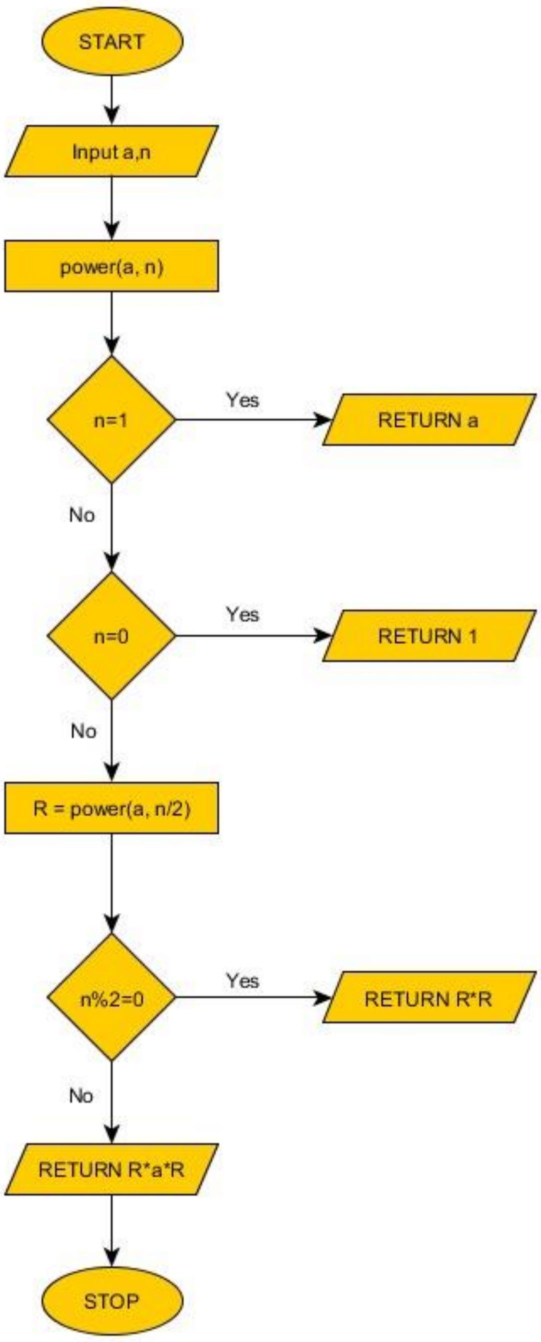


Figure 5: Fast Exponentiation Flowchart

3. VIT Antivirus

The final module of the project is a dictionary-based antivirus that not only helps us scan a file for its cleanliness but also provides the MD5 hash of the scanned file. The antivirus was built from scratch using the Microsoft Visual Studio 2015 RC environment and the .NET Framework 4.5. The antivirus is a very rudimentary one devoid of features like quarantine and full system scanning. A user-friendly GUI has been designed behind which the source code for the software is written. The source code uses namespaces such as System.Security which provides the underlying structure of the common language runtime security system, including base classes for permissions, and System.Security.Cryptography which allows us to work with hashes such as MD5, SHA and RACE. For the lifetime of this project, we will be only using the MD5 hash. After scanning a file, the software retrieves the embedded MD5 signature from the file. It then runs through a pre-made dictionary and checks if the retrieved hash code matches with any one of the codes inscribed in the dictionary. To do this, the dictionary used has been filled with virus signatures that were obtained from the database at www.virusshare.com.

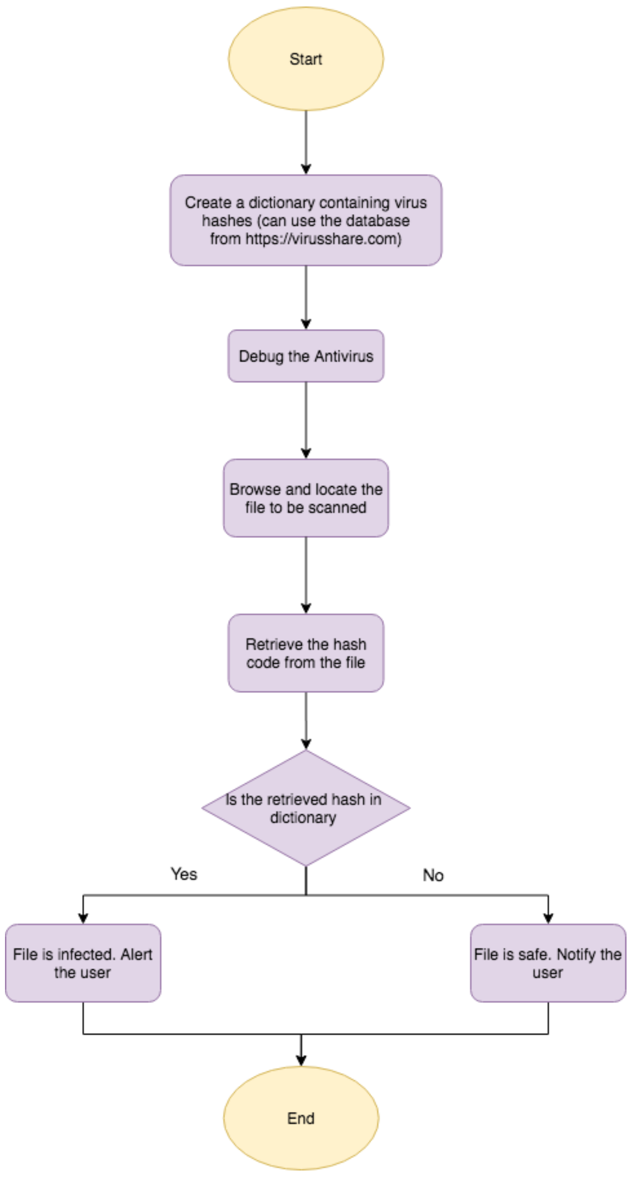

Start
Create a dictionary containing virus hashes (can use the database from https://virusshare.com)
Debug the Antivirus
Browse and locate the file to be scanned
Retrieve the hash code from the file
Is the retrieved hash in dictionary
Yes
No
File is infected. Alert the user
File is safe. Notify the user
End

Figure 6: VIT Antivirus working flowchart

4.2 CONSTRAINTS, ALTERNATIVES AND TRADEOFFS

Constraints

Following are the constraints that were experienced during the journey of this project:

- Due to expensive costs and the ongoing pandemic, actual resource-constrained devices could not be tested.
- Since resource-constrained devices are the target of this project, there was a complexity constraint that was followed during the making of the antivirus in order to keep it simple and user-friendly.

Alternatives

Following are the various alternatives that could be used:

- Instead of RSA, some other security algorithm that relies heavily on mathematical operations could be used for optimization.
- Different frameworks (Java, Mono, Ceylon, .NET Core, Jabaco) could have been used for the construction of the antivirus.
- ClamAV is an open source GPL anti-virus that provides a virus database as well. It could be used instead of VirusShare.

5. SCHEDULE, TASKS AND MILESTONES

5.1 SCHEDULING

For the proper completion of any project on time, a systematic task scheduling is necessary. Without a proper timeline, deadlines keep on getting delayed and the cost of completion also increases. The following figure properly demonstrates our working timeline and time management.

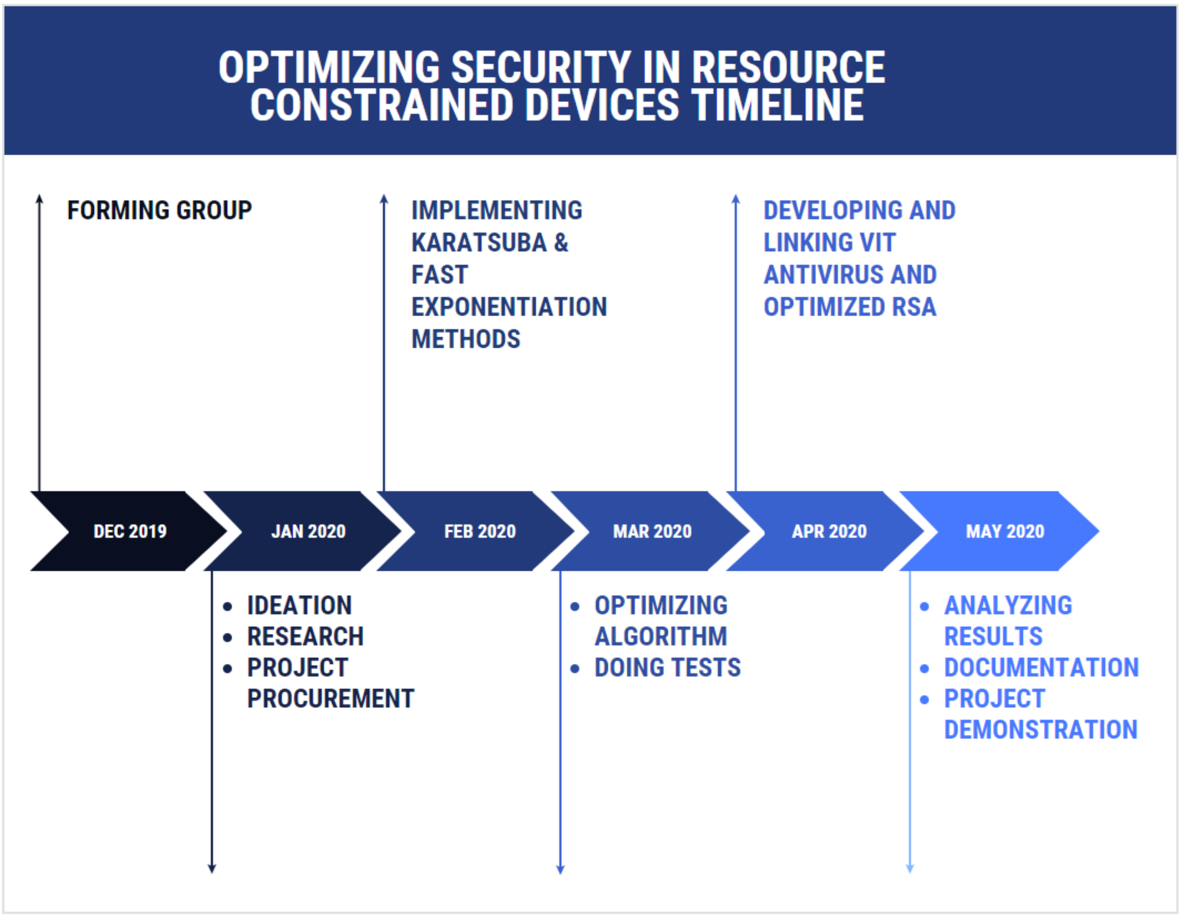


Figure 7: Scheduling of the project

5.2 TASKS

The completion of the project depends on the following major tasks:

1. Discussion among teammates: Proper discussion with teammates needs to be done to avoid any future disputes.

2. Discussion with the guide: Discuss the topic with the guide to check whether it is feasible.

3. Choosing the best algorithm suited for our project: Comparing different Security Algorithms such as AES, DES, RSA, etc. and choosing the right one for our project.

4. Reading research papers: Gaining knowledge for the project from previously done works.

5. Implementing Modules of the project: Implementing Karatsuba Method for fast multiplication and Fast Exponentiation Method.

6. Optimizing the algorithm: Implementing the methods chosen for optimization into the RSA algorithm.

7. Doing tests and getting results: Comparing the optimized algorithm with the aboriginal algorithm, removing errors, and noting down the results obtained.

8. Developing Antivirus: Developing the VIT Antivirus as another module for our project implementation.

9. Analyzing the results and reaching the conclusion: Comparing the overall security with the existing work, analyzing results, and reaching the conclusion.

10. Preparing PPT and report: Preparing a comprehensive report and an elaborate presentation for the review.

11. Presenting the presentation: The schedule and timeline for each task is given in the Gantt chart

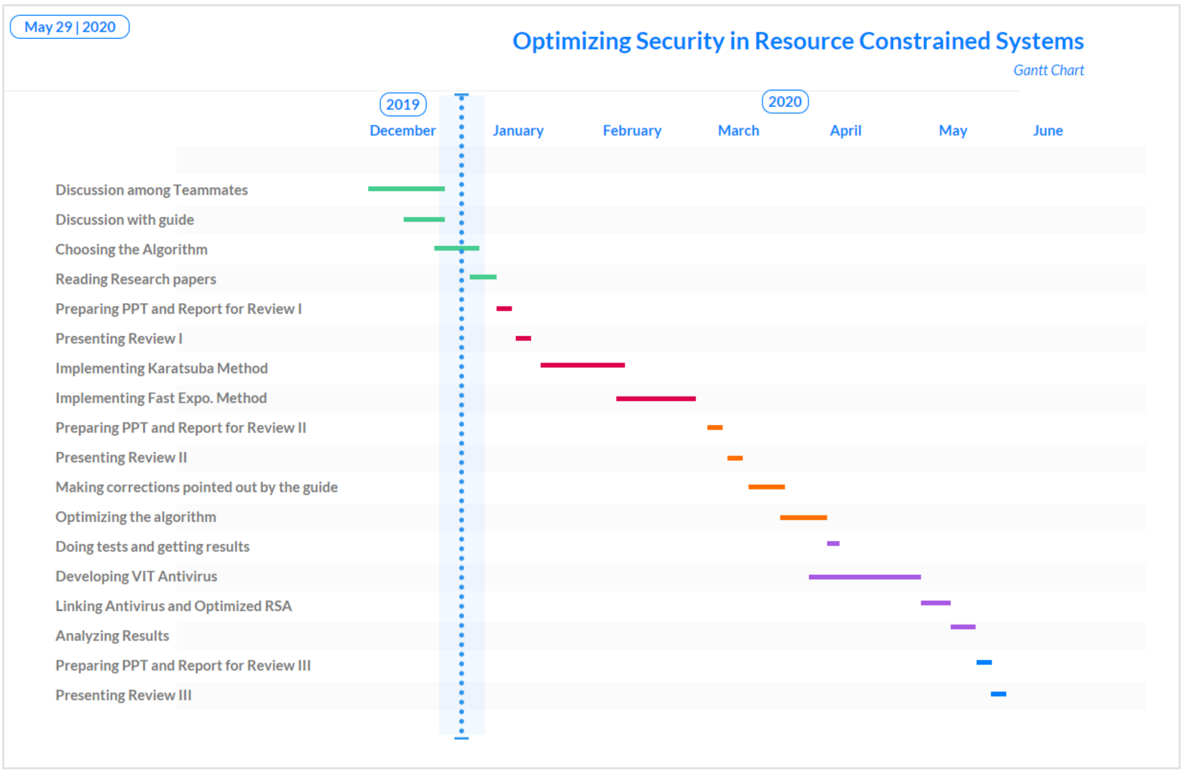


Figure 8: Gantt Chart of the project

5.3 MILESTONES

Major Milestones achieved during the journey of the project:

1. Implementation of the Karatsuba method for fast multiplication.
2. Implementation of the Fast Exponentiation method.
3. Optimization of the RSA algorithm using the implementations of the said methods.
4. In-house development of the antivirus called ‘VIT Antivirus’.
5. Successful linking of the VIT Antivirus with the Optimized RSA algorithm.

6. PROJECT DEMONSTRATION

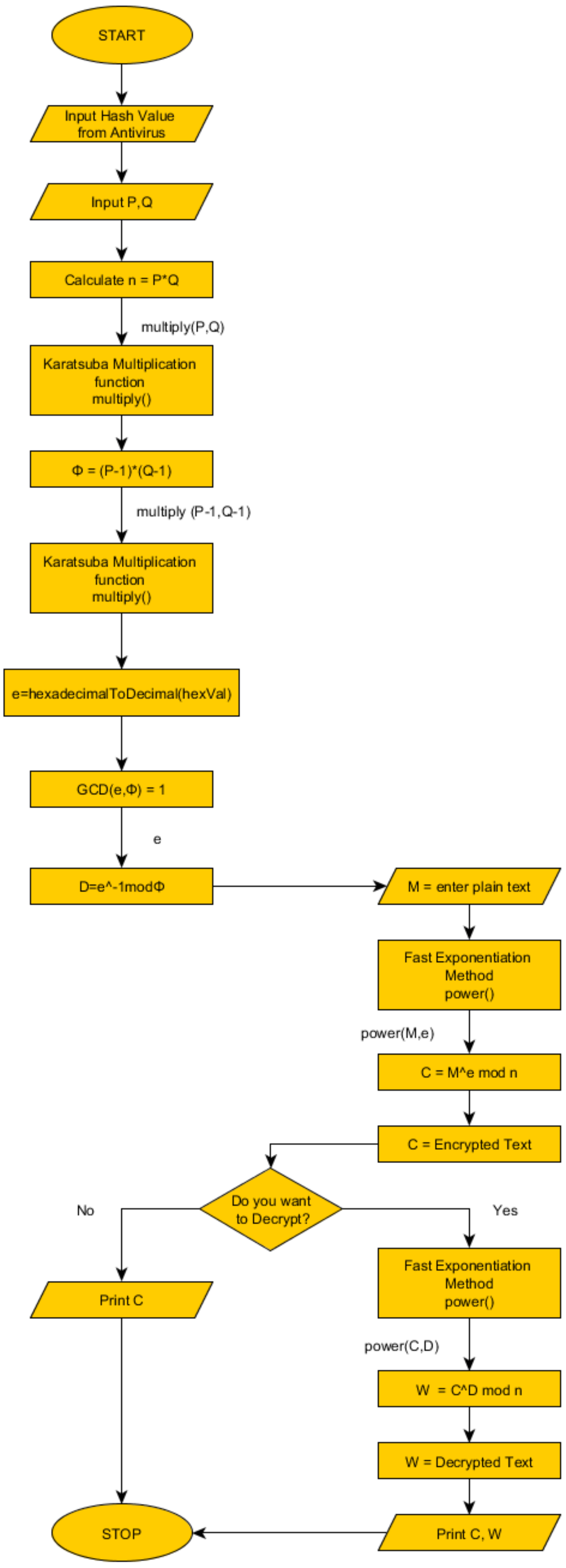
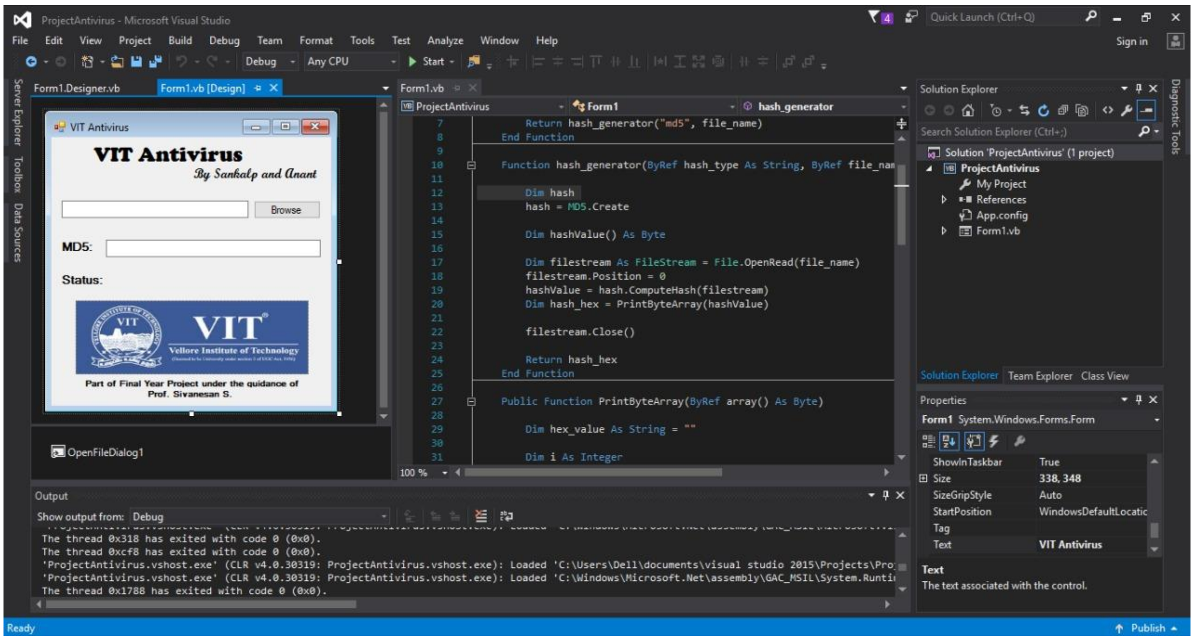


Figure 9: Project Flowchart

We start off by scanning a file (which is meant to be sent as a part of the public key) using the VIT Antivirus. The antivirus software extracts the MD5 hash signature from the file and searches it in the pre-defined dictionary. If the hash is present in the dictionary, the file is deemed as infected. Otherwise, a "Clean!" message pops up.

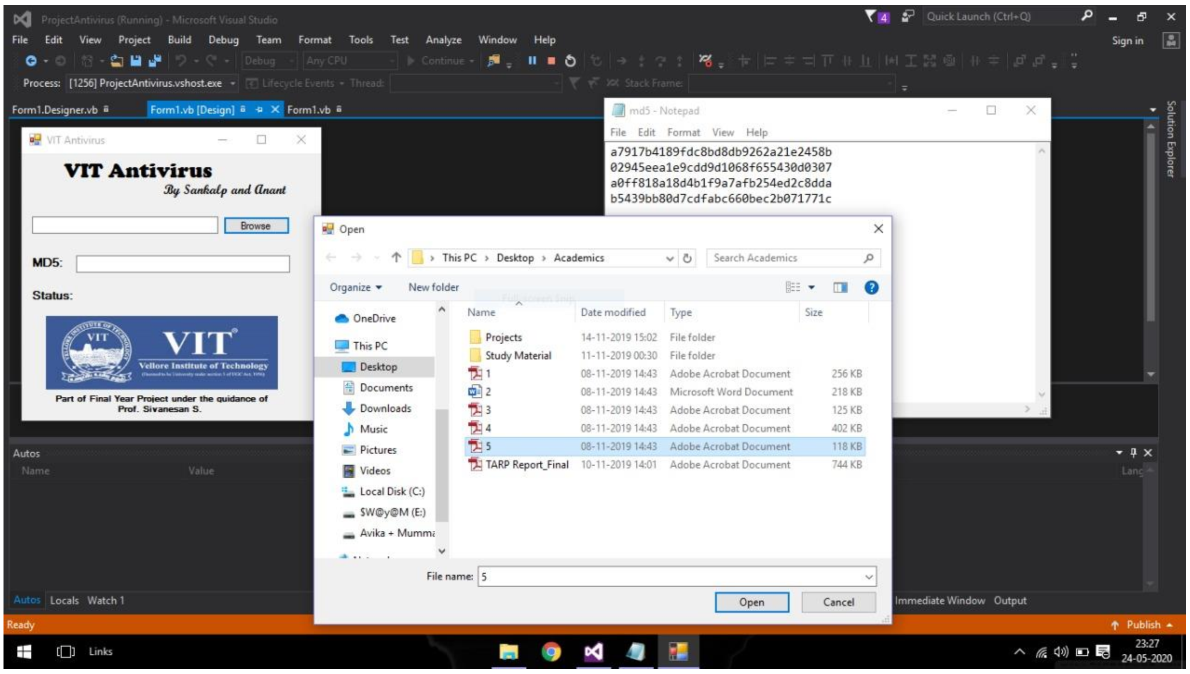


```
        Return hash_generator("md5", file_name)
    End Function

    Function hash_generator(ByRef hash_type As String, ByRef file_nam

        Dim hash
        hash = MD5.Create

        Dim hashValue() As Byte

        Dim filestream As FileStream = File.OpenRead(file_name)
        filestream.Position = 0
        hashValue = hash.ComputeHash(filestream)
        Dim hash_hex = PrintByteArray(hashValue)

        filestream.Close()

        Return hash_hex
    End Function

    Public Function PrintByteArray(ByRef array() As Byte)

        Dim hex_value As String = ""

        Dim i As Integer
```

```
The thread 0x318 has exited with code 0 (0x0).
The thread 0xcf8 has exited with code 0 (0x0).
'ProjectAntivirus.vshost.exe' (CLR v4.0.30319: ProjectAntivirus.vshost.exe): Loaded 'C:\Users\Dell\documents\visual studio 2015\Projects\Pro
'ProjectAntivirus.vshost.exe' (CLR v4.0.30319: ProjectAntivirus.vshost.exe): Loaded 'C:\Windows\Microsoft.Net\assembly\GAC_MSIL\System.Runti
The thread 0x1788 has exited with code 0 (0x0).
```

ProjectAntivirus - Microsoft Visual Studio
VIT Antivirus
By Sankalp and Anant
Browse
MD5:
Status:
VIT
Vellore Institute of Technology
Part of Final Year Project under the guidance of Prof. Sivanesan S.
OpenFileDialog1
Solution Explorer
Solution 'ProjectAntivirus' (1 project)
ProjectAntivirus
My Project
References
App.config
Form1.vb
Properties
Form1 System.Windows.Forms.Form
Text
The text associated with the control.
Ready

Figure 10: MS Visual Studio Workbench



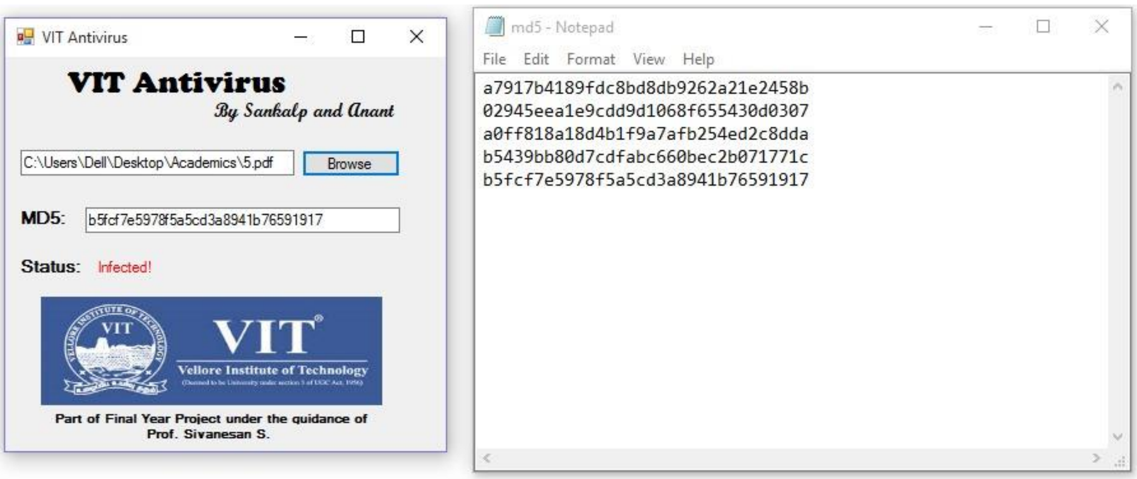
```
a7917b4189fdc8bd8db9262a21e2458b
02945eea1e9cdd9d1068f655430d0307
a0ff818a18d4b1f9a7afb254ed2c8dda
b5439bb80d7cdfabc660bec2b071771c
```

Name	Date modified	Type	Size
Projects	14-11-2019 15:02	File folder	
Study Material	11-11-2019 00:30	File folder	
1	08-11-2019 14:43	Adobe Acrobat Document	256 KB
2	08-11-2019 14:43	Microsoft Word Document	218 KB
3	08-11-2019 14:43	Adobe Acrobat Document	125 KB
4	08-11-2019 14:43	Adobe Acrobat Document	402 KB
5	08-11-2019 14:43	Adobe Acrobat Document	118 KB
TARP Report_Final	10-11-2019 14:01	Adobe Acrobat Document	744 KB

File name: 5

Figure 11: Selecting a file for scanning

In figure 10, we select a file from the local system to check if it is safe or not.


VIT Antivirus
VIT Antivirus
By Sankalp and Anant
Browse
VIT
Vellore Institute of Technology
Part of Final Year Project under the guidance of Prof. Sivanesan S.
md5 - Notepad
File Edit Format View Help

C:\Users\Dell\Desktop\Academics\5.pdf

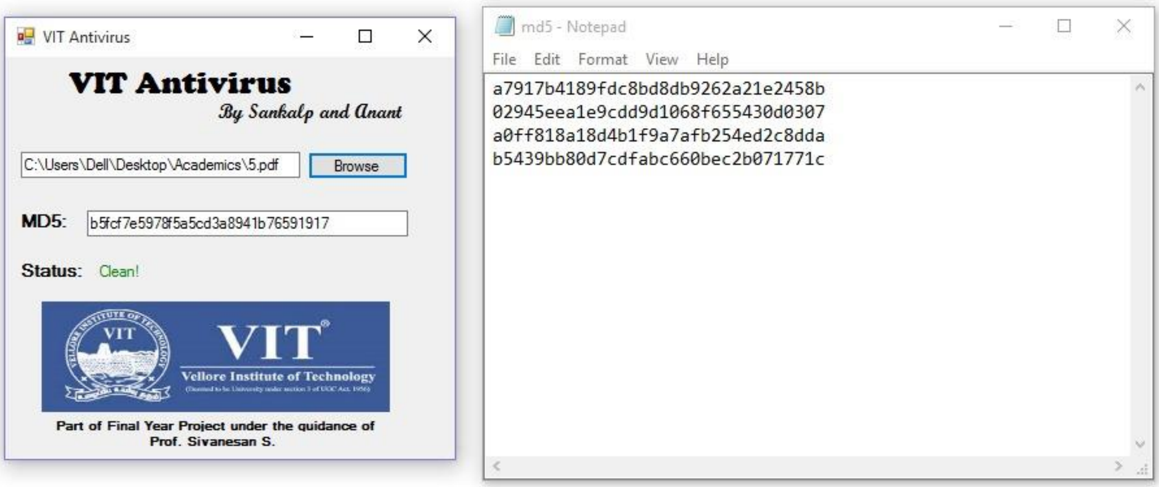
MD5: b5fcf7e5978f5a5cd3a8941b76591917

Status: Infected!

```
a7917b4189fdc8bd8db9262a21e2458b
02945eea1e9cdd9d1068f655430d0307
a0ff818a18d4b1f9a7afb254ed2c8dda
b5439bb80d7cdfabc660bec2b071771c
b5fcf7e5978f5a5cd3a8941b76591917
```

Figure 12: File signature found in dictionary

In figure 11, since the file’s hash signature exists in the dictionary, it is labelled as ‘Infected’.


VIT Antivirus
VIT Antivirus
By Sankalp and Anant
Browse
VIT
Vellore Institute of Technology
Part of Final Year Project under the guidance of Prof. Sivanesan S.
md5 - Notepad
File Edit Format View Help

C:\Users\Dell\Desktop\Academics\5.pdf

MD5: b5fcf7e5978f5a5cd3a8941b76591917

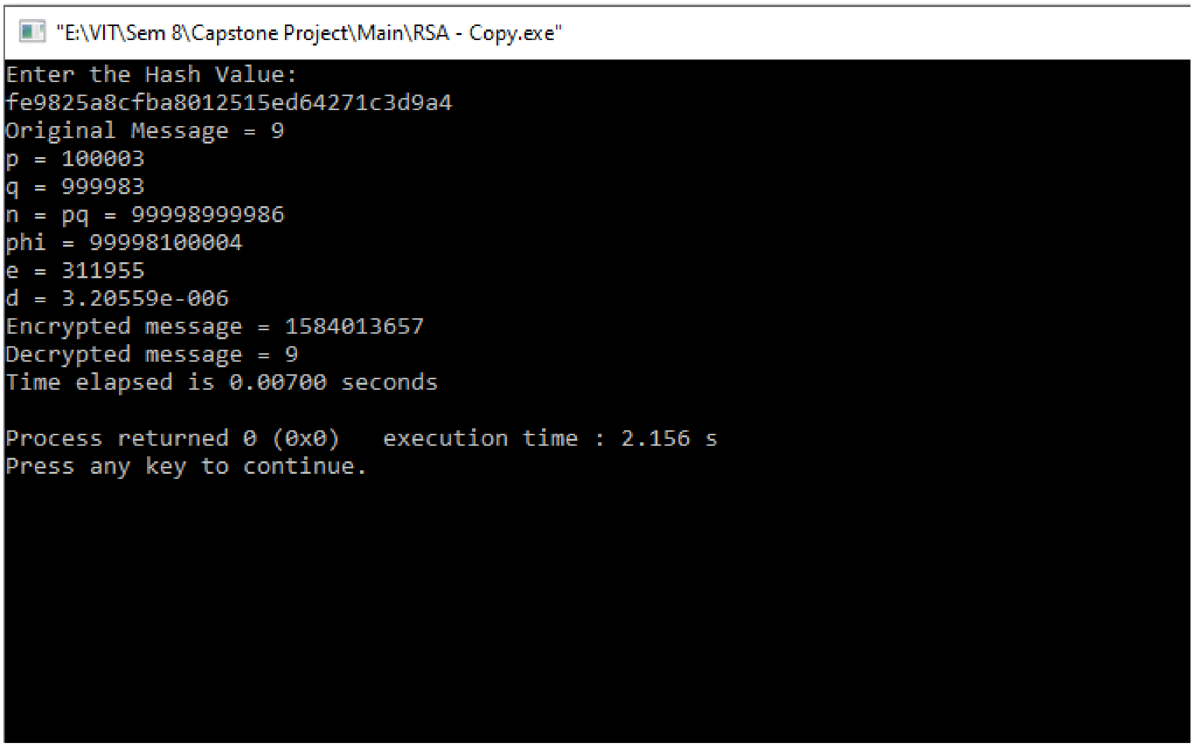
Status: Clean!

```
a7917b4189fdc8bd8db9262a21e2458b
02945eea1e9cdd9d1068f655430d0307
a0ff818a18d4b1f9a7afb254ed2c8dda
b5439bb80d7cdfabc660bec2b071771c
```

Figure 13: File becomes clean

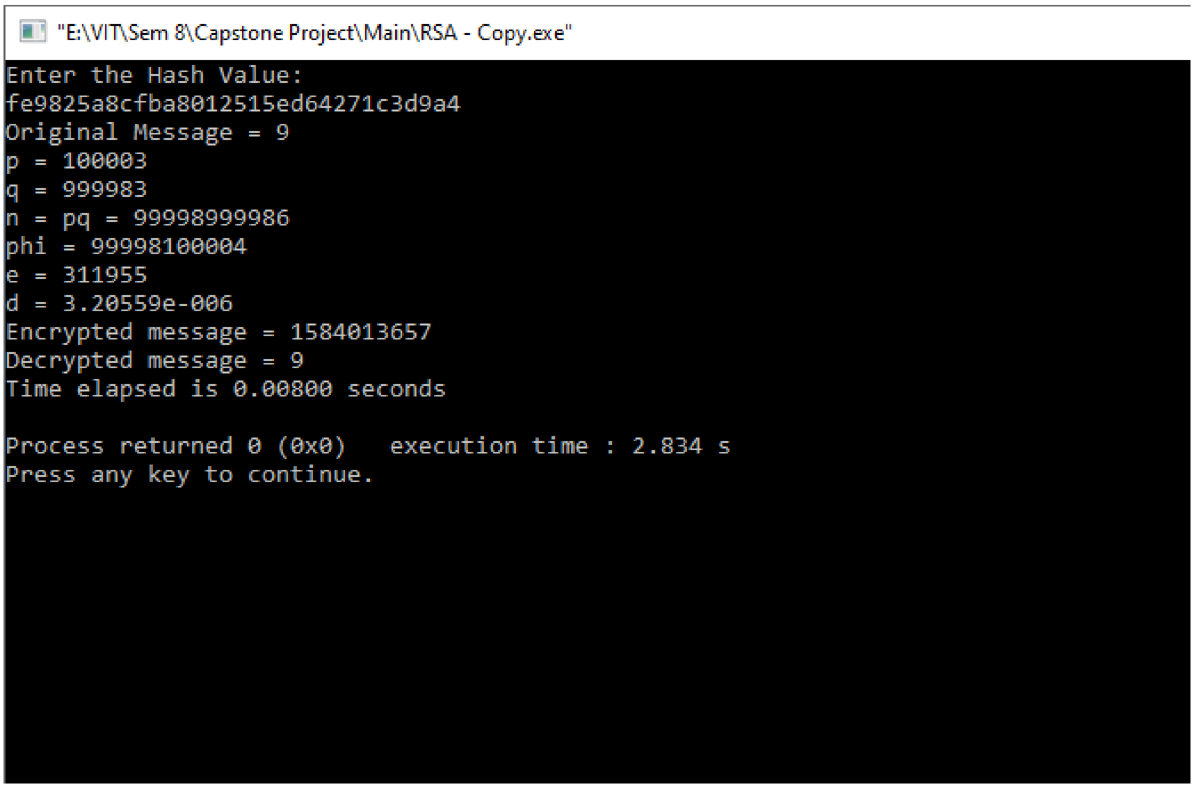
Now, the same hash signature has been added to the dictionary. This makes the Antivirus label the file as ‘Clean’.

Now, for the RSA part, similar to the process above, we scan a file and retrieve a hash code from it. The retrieved code is then used as an input for the RSA algorithm as shown below.



```
"E:\VIT\Sem 8\Capstone Project\Main\RSA - Copy.exe"
Enter the Hash Value:
fe9825a8cfba8012515ed64271c3d9a4
Original Message = 9
p = 100003
q = 999983
n = pq = 99998999986
phi = 99998100004
e = 311955
d = 3.20559e-006
Encrypted message = 1584013657
Decrypted message = 9
Time elapsed is 0.00700 seconds

Process returned 0 (0x0)   execution time : 2.156 s
Press any key to continue.
```

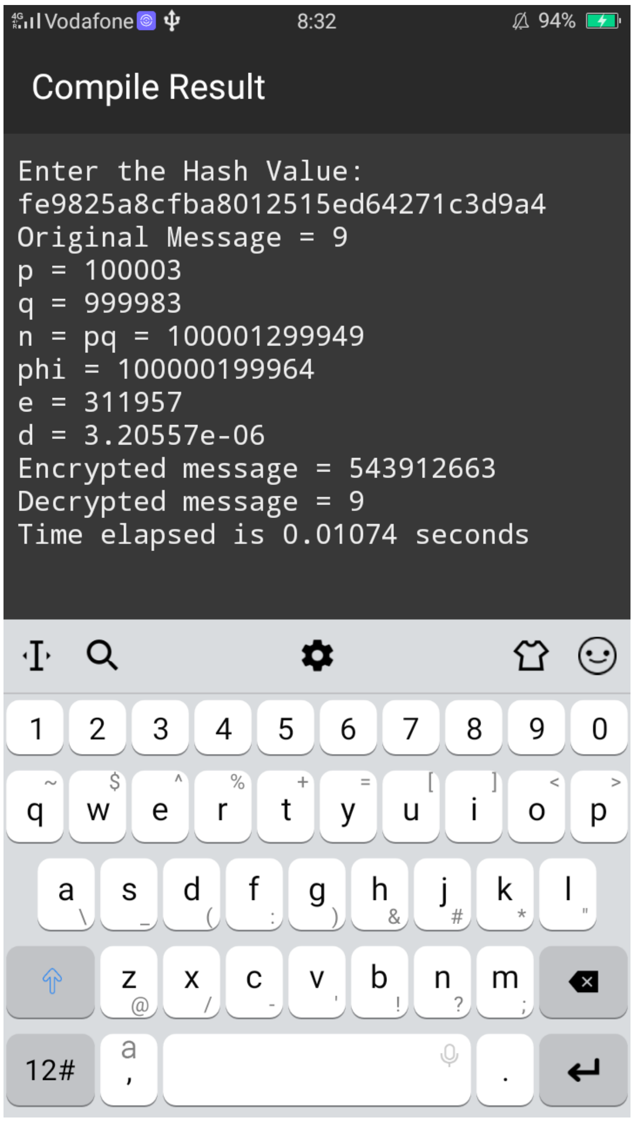
Figure 14: PC implementation of Optimized RSA 1



```
"E:\VIT\Sem 8\Capstone Project\Main\RSA - Copy.exe"
Enter the Hash Value:
fe9825a8cfba8012515ed64271c3d9a4
Original Message = 9
p = 100003
q = 999983
n = pq = 99998999986
phi = 99998100004
e = 311955
d = 3.20559e-006
Encrypted message = 1584013657
Decrypted message = 9
Time elapsed is 0.00800 seconds

Process returned 0 (0x0)   execution time : 2.834 s
Press any key to continue.
```

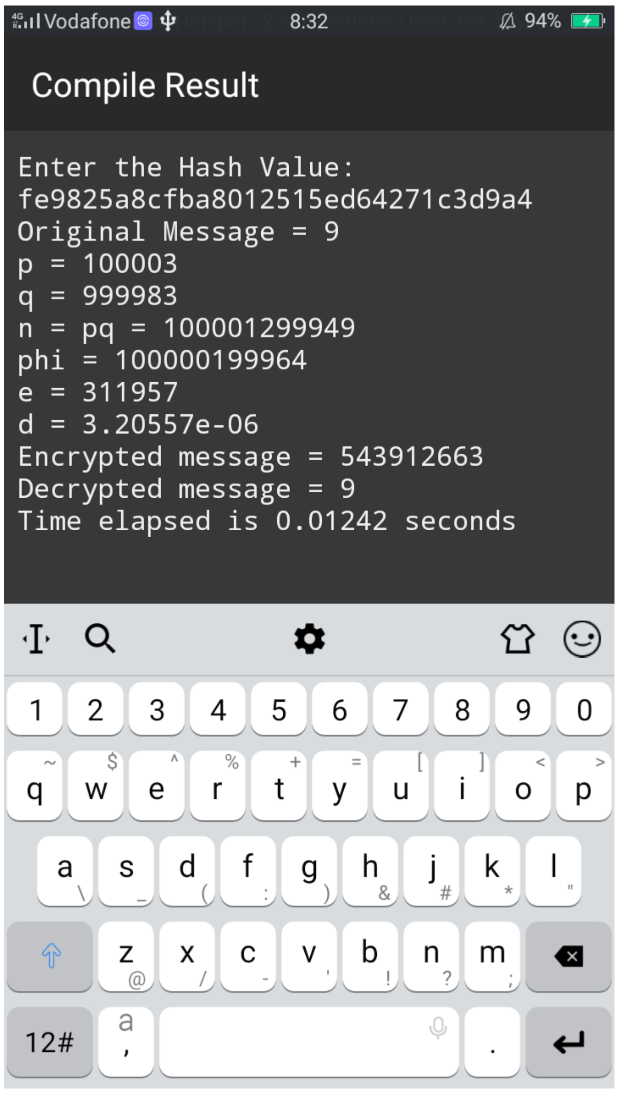
Figure 15: PC implementation of Optimized RSA 2

To check the working of the same in a resource-constrained device, a low-spec smartphone is used next.



```
Enter the Hash Value:
fe9825a8cfba8012515ed64271c3d9a4
Original Message = 9
p = 100003
q = 999983
n = pq = 100001299949
phi = 100000199964
e = 311957
d = 3.20557e-06
Encrypted message = 543912663
Decrypted message = 9
Time elapsed is 0.01074 seconds
```

Figure 16: Smartphone implementation of Optimized RSA 1



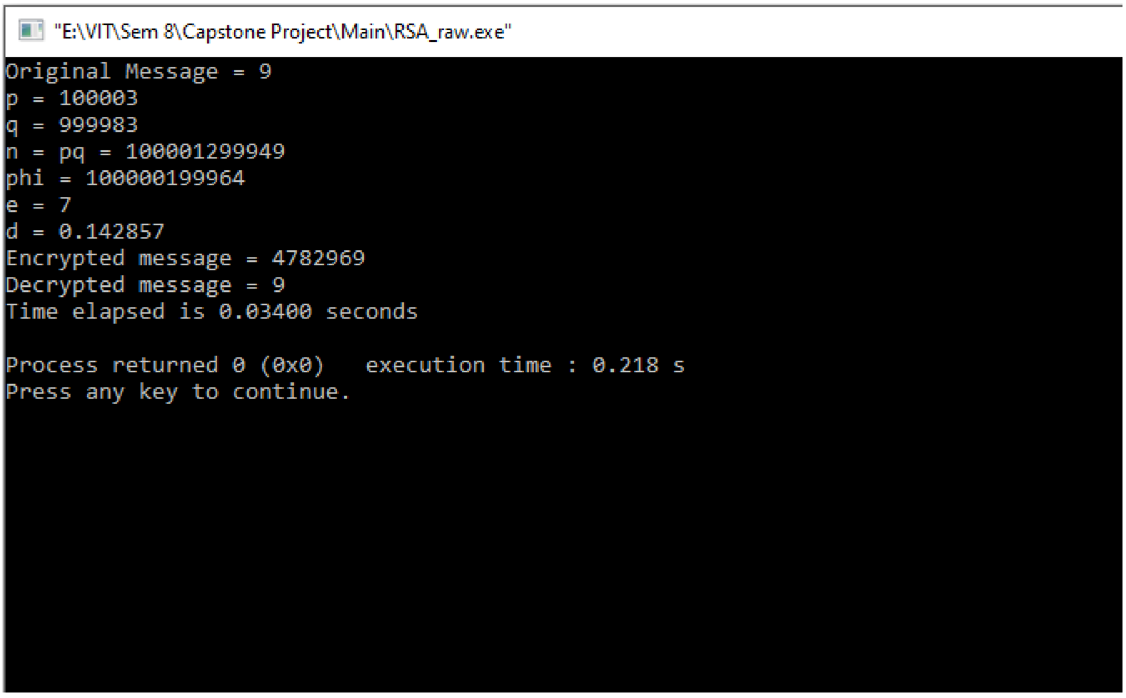
```
Enter the Hash Value:
fe9825a8cfba8012515ed64271c3d9a4
Original Message = 9
p = 100003
q = 999983
n = pq = 100001299949
phi = 100000199964
e = 311957
d = 3.20557e-06
Encrypted message = 543912663
Decrypted message = 9
Time elapsed is 0.01242 seconds
```

Figure 17: Smartphone implementation of Optimized RSA 2

Thus, we receive very efficient time metrics, even in a resource-constrained device, thats relatively very low when compared to the time metrics of the unmodified RSA algorithm, the comparison of which we will see in the next chapter.

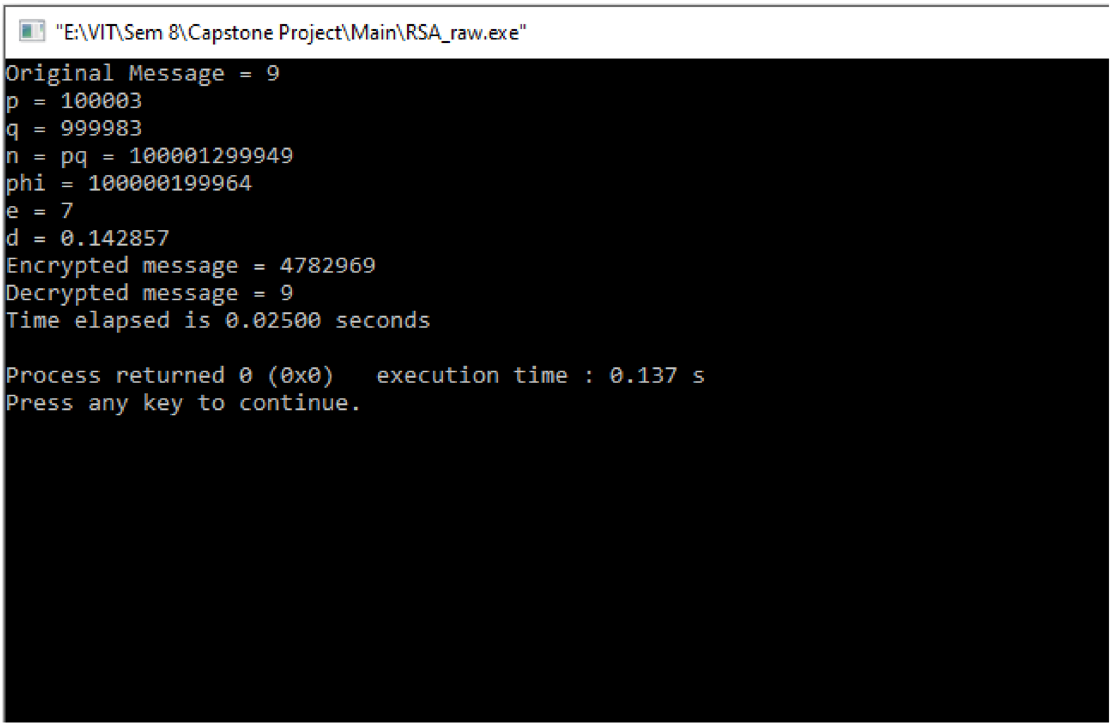
7. RESULTS AND DISCUSSION

In the previous chapter, we have seen the screenshots of the compile time for the optimized version of the algorithm. Following are the same for the original RSA algorithm.



```
"E:\VIT\Sem 8\Capstone Project\Main\RSA_raw.exe"
Original Message = 9
p = 100003
q = 999983
n = pq = 100001299949
phi = 100000199964
e = 7
d = 0.142857
Encrypted message = 4782969
Decrypted message = 9
Time elapsed is 0.03400 seconds

Process returned 0 (0x0)   execution time : 0.218 s
Press any key to continue.
```

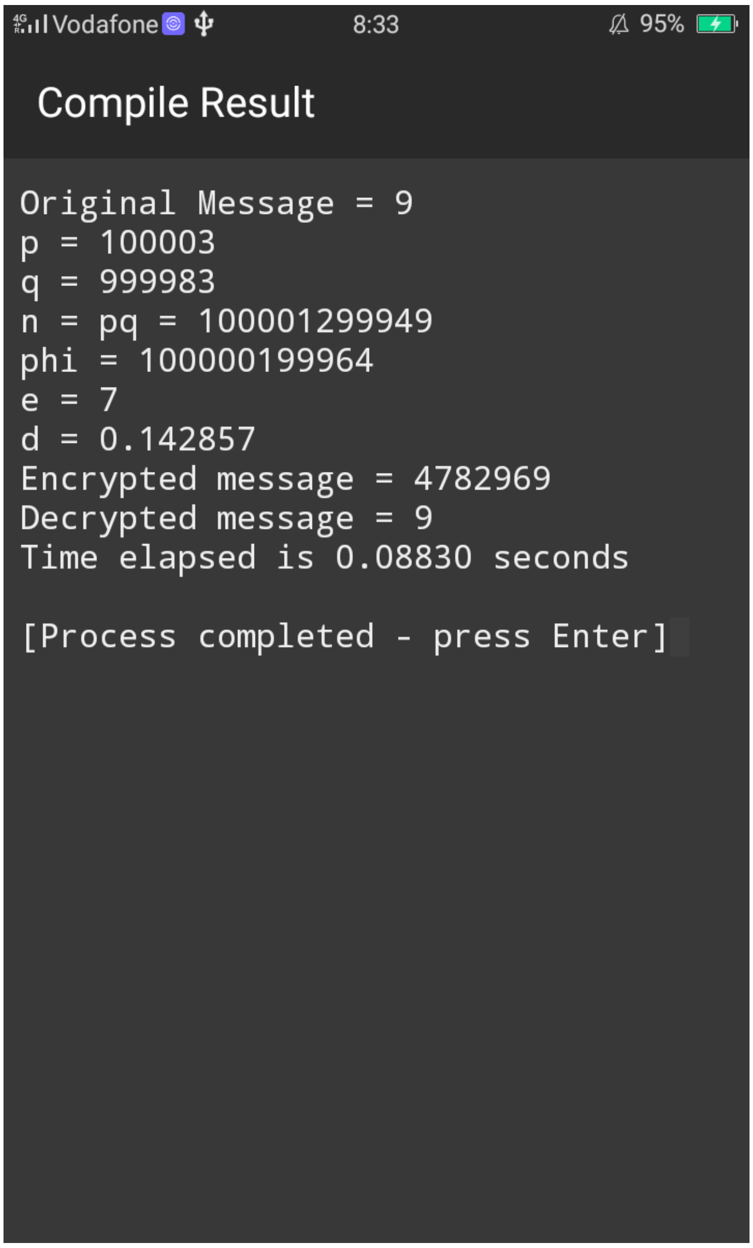
Figure 18: PC implementation of normal RSA 1



```
"E:\VIT\Sem 8\Capstone Project\Main\RSA_raw.exe"
Original Message = 9
p = 100003
q = 999983
n = pq = 100001299949
phi = 100000199964
e = 7
d = 0.142857
Encrypted message = 4782969
Decrypted message = 9
Time elapsed is 0.02500 seconds

Process returned 0 (0x0)   execution time : 0.137 s
Press any key to continue.
```

Figure 19: PC implementation of normal RSA 2

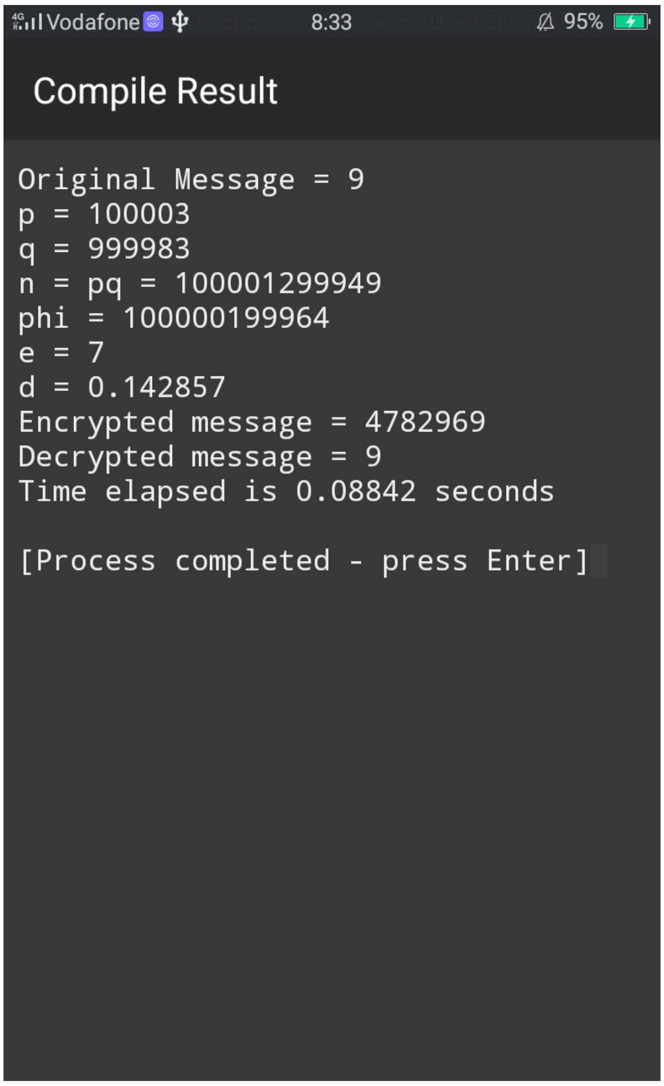
Implementation in a resource-constrained device (low-end smartphone):



Compile Result

```
Original Message = 9
p = 100003
q = 999983
n = pq = 100001299949
phi = 100000199964
e = 7
d = 0.142857
Encrypted message = 4782969
Decrypted message = 9
Time elapsed is 0.08830 seconds

[Process completed - press Enter]
```

Figure 20: Smartphone implementation of normal RSA 1



```
Original Message = 9
p = 100003
q = 999983
n = pq = 100001299949
phi = 100000199964
e = 7
d = 0.142857
Encrypted message = 4782969
Decrypted message = 9
Time elapsed is 0.08842 seconds

[Process completed - press Enter]
```

Figure 21: Smartphone implementation of normal RSA 2

Clearly, there is a considerable difference between the compile times of the original RSA and the optimized RSA. The following table displays the figures for the comparison.

Table 1: Comparison of different RSA implementations on different devices

Device	Algorithm	Average Time Taken (s)	Result
Laptop	Normal RSA	0.029	Optimized RSA is 3-4 times faster than Normal RSA Algorithm.
	Optimized RSA	0.007	
Phone	Normal RSA	0.08830	Optimized RSA is 8-9 times faster than Normal RSA Algorithm
	Optimized RSA	0.01158	

8. SUMMARY

The goals and objectives of the project have been discussed. The design approach followed is documented and explained. The working of the mathematical techniques used has been elaborated. The namespaces, libraries and frameworks used have been mentioned with their functioning. The overall working structure of the two modules is illustrated with the help of comprehensive flowcharts. A Gantt Chart has been pinned to showcase the scheduling and tasks.

Resource-Constrained devices have become a mainstay and they are surely going to be mainstream in the future as well. This calls for the continuous growth of the security industry, especially in areas like these, in order for us to live in a world where privacy and security are not considered luxuries and the safety of all devices is paramount.

9. REFERENCES

[1] R.L. Rivest, A. Shamir, and L. Adleman. (1978). A journal on A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.

[2] A. Mollin, Richard, Chapman (2000). A book on RSA and Public-key Cryptography. Hall/CRC.

[3] Stalling, William. (2004). Cryptography and Network Security, Principle and Practive. Prentice Hall.

[4] Gulen U, Alkhodary A, Baktir S. "Implementing RSA for Wireless Sensor Nodes. *Sensors" (Basel)*. 2019;19(13):2864. Published 2019 Jun 27. doi:10.3390/s19132864

[5] Manish Kumar Goyal, Shiv Karan Meghwal. "Analysis of GF (2m) Multiplication Algorithm: Classic Method v/s Karatsuba-Ofman Multiplication Method", International Journal on Recent Innovation Trends on Computing and Communication 2231-8169

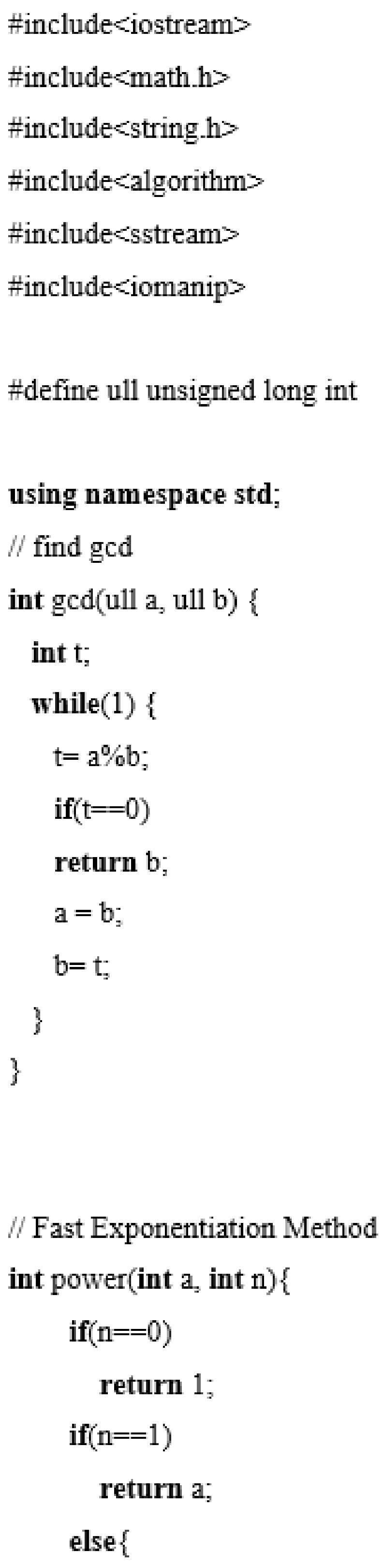
[6] Peter Kormerup, "High-Radix Modular Multiplication for Cryptosystems", IEEE 1063-6889/93 pp. 277-283

[7] VirusShare. http://www.virusshare.com/

APPENDIX A

Project Code:

Optimized_RSA.cpp

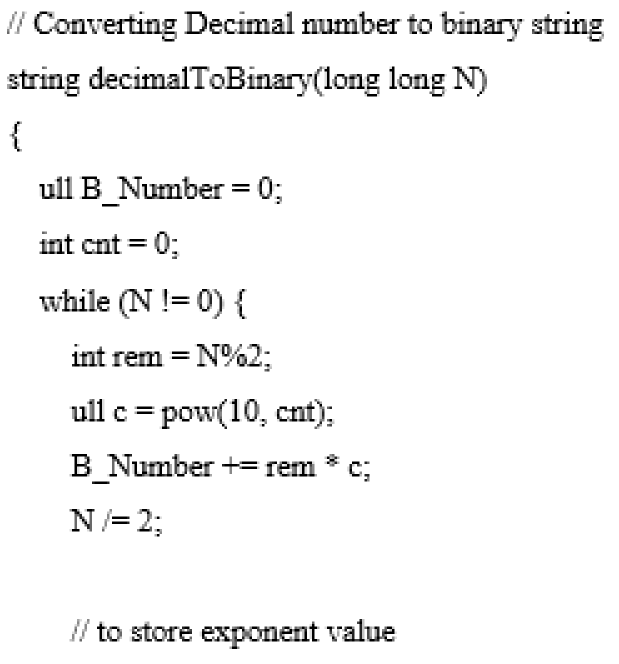


```
#include<iostream>
#include<math.h>
#include<string.h>
#include<algorithm>
#include<sstream>
#include<iomanip>

#define ull unsigned long int

using namespace std;
// find gcd
int gcd(ull a, ull b) {
  int t;
  while(1) {
    t= a%b;
    if(t==0)
    return b;
    a = b;
    b= t;
  }
}



// Fast Exponentiation Method
int power(int a, int n){
      if(n==0)
         return 1;
      if(n==1)
         return a;
      else{

           return R*a*R;
```

```
        }
}

// Equalizing String Lengths
int equalise_length(string &s1, string &s2) {

    int l1=s1.size(), l2=s2.size();

    if (l1<l2)
    {
        for(int i=0; i<l2-l1; i++)

        s1='0'+s1;
    }

    else if (l2<l1)
   {
        for(int i=0; i<l1-l2; i++)

        s2='0'+s2;

        return l1;
   }

   return l2;

}

// Adding Strings After Equalizing their Lengths
string add (string a, string b) {

   int l=equalise_length(a,b),ai,bi,carry=0,ci;
   string c;
```

```
    for (int i= 1; i>=0; i--) {

        ai=a. +'l0';
        bi=b.+'l0';

        ci= (ai^bi^carry);

        c=(char)(ci+'0')+c;

        carry = (ai&bi) | (ai&carry) |(bi&carry);
    }

    if (carry)
    {
        c = '1' + c;
    }


    return c;

}


// Karatsuba Method for Fast Multiplication
long multiply(string a, string b)

{

    int l = equalise_length(a,b);

    if(l==0)
        return 0;

    if (l==1)
```

```
        return (int)(a[0]-'0')&(int)(b[0]-'0');

    int l1 = l/2;
    int l2= (l-l1);

    string a1=a.substr(0,l1);
    string a2=a.substr(l1,l2);

    string b1=b.substr(0,l1);
    string b2=b.substr(l1,l2);

    long long a1b1= multiply(a1, b1);
    long long a2b2= multiply(a2, b2);

    long long P= multiply(add(a1,a2),add(b1,b2));

    return a1b1*(1<<(2*l2)) + (P - a1b1 - a2b2)*(1<<l2) + a2b2;

}
```

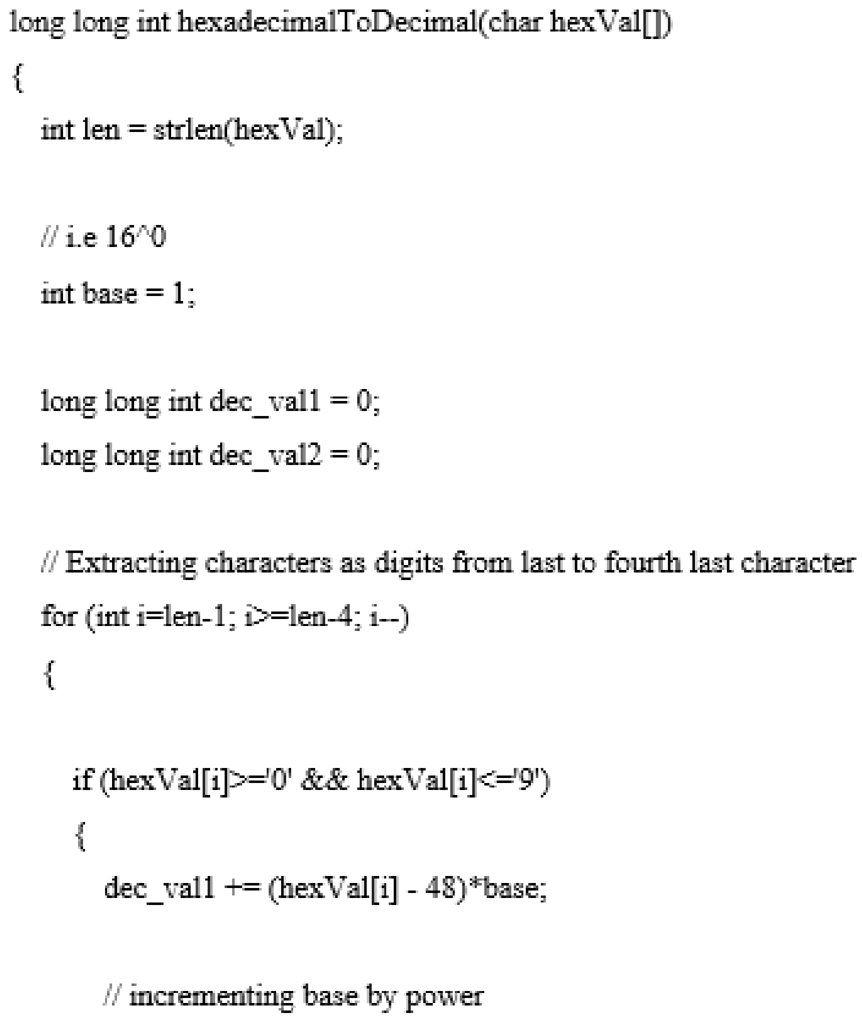


```
// Converting Decimal number to binary string
string decimalToBinary(long long N)
{
    ull B_Number = 0;
    int cnt = 0;
    while (N != 0) {
        int rem = N%2;
        ull c = pow(10, cnt);
        B_Number += rem * c;
        N /= 2;

        // to store exponent value
```

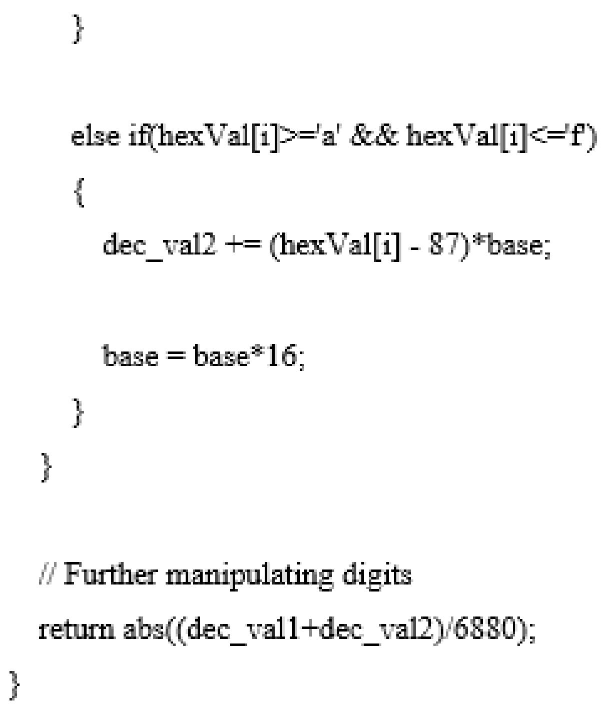
```
        cnt++;
```

```
    }

    stringstream ss;
    ss<<B_No;

    string s;

    ss>>s;

    return s;
}
```

```
// Converting the Hash Value obtained from Antivirus to Decimal in a custom format
```



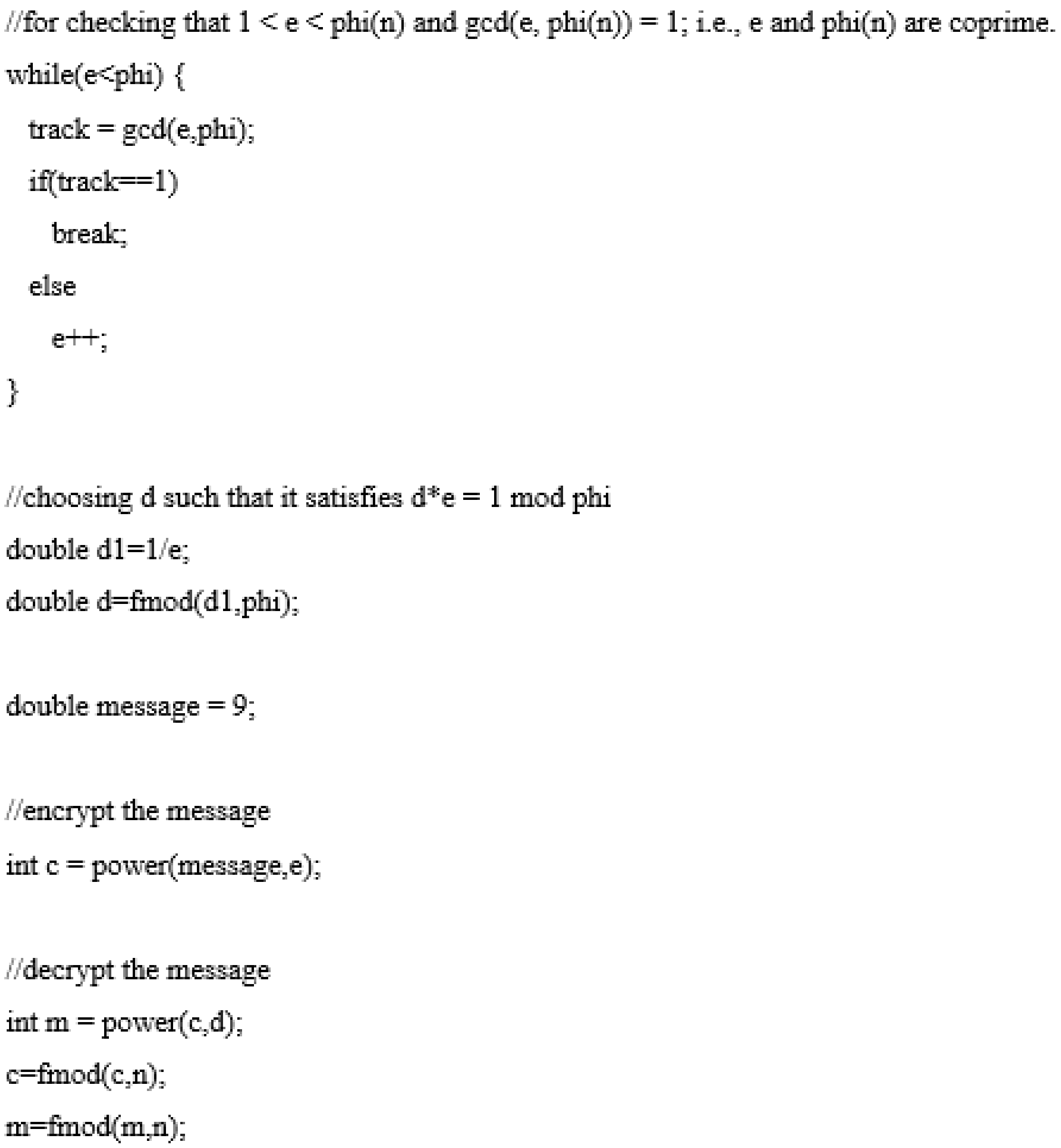
```
long long int hexadecimalToDecimal(char hexVal[])
{
    int len = strlen(hexVal);

    // i.e 16^0
    int base = 1;

    long long int dec_val1 = 0;
    long long int dec_val2 = 0;

    // Extracting characters as digits from last to fourth last character
    for (int i=len-1; i>=len-4; i--)
    {

        if (hexVal[i]>='0' && hexVal[i]<='9')
        {
            dec_val1 += (hexVal[i] - 48)*base;

            // incrementing base by power
        base = base * 16;
    }

    else if (hexVal[i]>='A' && hexVal[i]<='F')
    {
        dec_val1 += (hexVal[i] - 55)*base;

        base = base*16;
    }
    else if(hexVal[i]>='a' && hexVal[i]<='f')
    {
        dec_val1 += (hexVal[i] - 87)*base;

        base = base*16;
    }
}


// Extracting characters as digits from 16th to 20th character from the end
for (int i=len-17; i>=len-21; i--)
{

    if (hexVal[i]>='0' && hexVal[i]<='9')
    {
        dec_val2 += (hexVal[i] - 48)*base;

        base = base * 16;
    }

    else if (hexVal[i]>='a' && hexVal[i]<='f' || hexVal[i]>='A' && hexVal[i]<='F')
    {
        dec_val2 += (hexVal[i] - 55)*base;

        base = base*16;
```



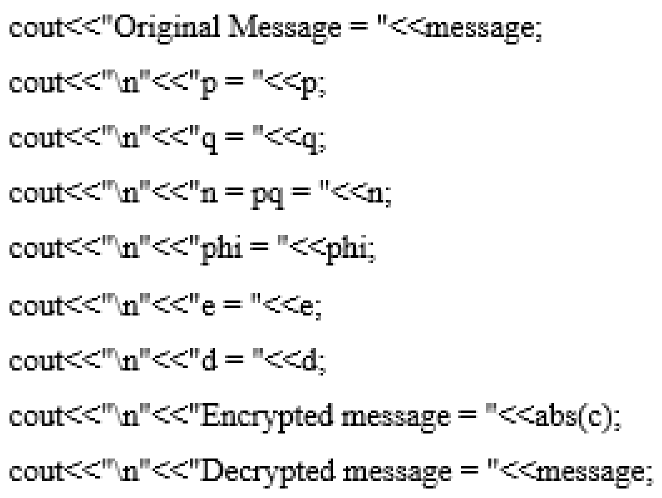
```
        }

        else if(hexVal[i]>='a' && hexVal[i]<='f')
        {
            dec_val2 += (hexVal[i] - 87)*base;

            base = base*16;
        }
    }

    // Further manipulating digits
    return abs((dec_val1+dec_val2)/6880);
}
```

```
int main() {



    char hexVal[32];


    cout<<"Enter the Hash Value: "<<endl;
    cin>>hexVal;
    double e=hexadecimalToDecimal(hexVal);



    clock_t tim;


    tim = clock();


    //2 random prime numbers
   long long p = 100003;
   long long q = 999983;


   string p1 = decimalToBinary(p);
   string q1 = decimalToBinary(q);
```

```
  long long n = multiply(p1,q1);
  double track;

  string p2 = decimalToBinary(p-1);
  string q2 = decimalToBinary(q-1);

  //Calculating phi(n)
  long long phi= multiply(p2,q2);
```



```
//for checking that 1 < e < phi(n) and gcd(e, phi(n)) = 1; i.e., e and phi(n) are coprime.
while(e<phi) {
   track = gcd(e,phi);
   if(track==1)
      break;
   else
      e++;
}

//choosing d such that it satisfies d*e = 1 mod phi
double d1=1/e;
double d=fmod(d1,phi);

double message = 9;

//encrypt the message
int c = power(message,e);

//decrypt the message
int m = power(c,d);
c=fmod(c,n);
m=fmod(m,n);
```



```
cout<<"Original Message = "<<message;
cout<<"\n"<<"p = "<<p;
cout<<"\n"<<"q = "<<q;
cout<<"\n"<<"n = pq = "<<n;
cout<<"\n"<<"phi = "<<phi;
cout<<"\n"<<"e = "<<e;
cout<<"\n"<<"d = "<<d;
cout<<"\n"<<"Encrypted message = "<<abs(c);
cout<<"\n"<<"Decrypted message = "<<message;
```

```
    tim = clock()-tim;

    double time_spent1 = 0.0;

    time_spent1 += (double)(tim) / CLOCKS_PER_SEC;

     cout<<endl;

    cout<<"Time elapsed is "<<fixed<<setprecision(5)<<time_spent1<<" seconds"<<endl;


   return 0;
 }
```

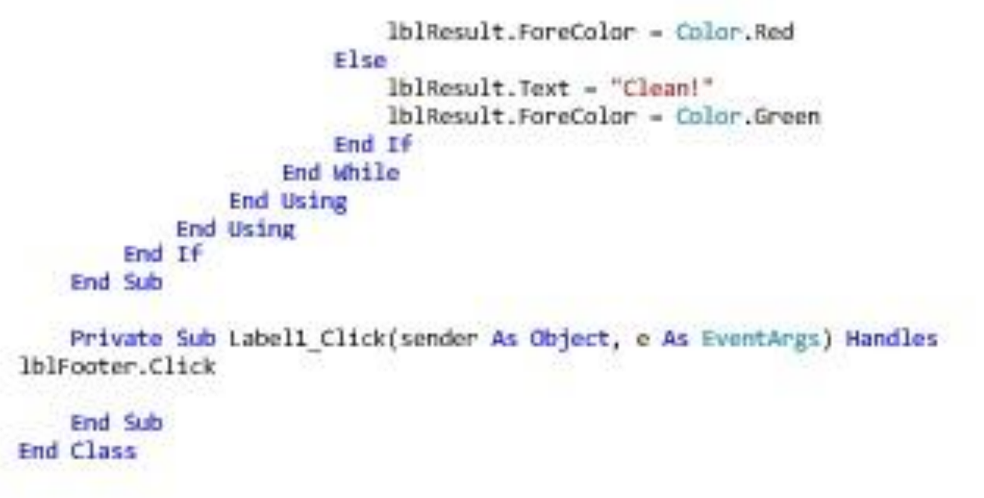
Form1.Designer.vb

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()>
Partial Class Form1
    Inherits System.Windows.Forms.Form

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()>
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        Try
            If disposing AndAlso components IsNot Nothing Then
                components.Dispose()
            End If
        Finally
            MyBase.Dispose(disposing)
        End Try
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()>
    Private Sub InitializeComponent()
        Dim resources As System.ComponentModel.ComponentResourceManager = New
System.ComponentModel.ComponentResourceManager(GetType(Form1))
        Me.lblTitle = New System.Windows.Forms.Label()
        Me.lblSig = New System.Windows.Forms.Label()
        Me.txtFilePath = New System.Windows.Forms.TextBox()
        Me.btnBrowse = New System.Windows.Forms.Button()
        Me.txtHash = New System.Windows.Forms.TextBox()
        Me.lblMD5 = New System.Windows.Forms.Label()
        Me.lblStatus = New System.Windows.Forms.Label()
        Me.lblResult = New System.Windows.Forms.Label()
        Me.vitLogo = New System.Windows.Forms.PictureBox()
        Me.OpenFileDialog1 = New System.Windows.Forms.OpenFileDialog()
        Me.lblFooter = New System.Windows.Forms.Label()
        Me.Label1 = New System.Windows.Forms.Label()
        CType(Me.vitLogo, System.ComponentModel.ISupportInitialize).BeginInit()
        Me.SuspendLayout()
        '
        'lblTitle
        '
        Me.lblTitle.AutoSize = True
        Me.lblTitle.Font = New System.Drawing.Font("Cooper Black", 18.0!,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.lblTitle.Location = New System.Drawing.Point(44, 6)
        Me.lblTitle.Name = "lblTitle"
        Me.lblTitle.Size = New System.Drawing.Size(183, 27)
        Me.lblTitle.TabIndex = 0
        Me.lblTitle.Text = "VIT Antivirus"
        '
        'lblSig
        '
        Me.lblSig.AutoSize = True
        Me.lblSig.Font = New System.Drawing.Font("Script MT Bold", 11.25!,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.lblSig.Location = New System.Drawing.Point(161, 33)
        Me.lblSig.Name = "lblSig"
        Me.lblSig.Size = New System.Drawing.Size(149, 18)
        Me.lblSig.TabIndex = 1
        Me.lblSig.Text = "By Sankalp and Anant"
        '
        'txtFilePath
        '
        Me.txtFilePath.Location = New System.Drawing.Point(12, 73)
        Me.txtFilePath.Name = "txtFilePath"
        Me.txtFilePath.Size = New System.Drawing.Size(215, 20)
        Me.txtFilePath.TabIndex = 2
        '
        'btnBrowse
        '
        Me.btnBrowse.Location = New System.Drawing.Point(233, 73)
        Me.btnBrowse.Name = "btnBrowse"
        Me.btnBrowse.Size = New System.Drawing.Size(77, 21)
        Me.btnBrowse.TabIndex = 3
        Me.btnBrowse.Text = "Browse"
        Me.btnBrowse.UseVisualStyleBackColor = True
        '
        'txtHash
        '
        Me.txtHash.Location = New System.Drawing.Point(63, 118)
        Me.txtHash.Name = "txtHash"
        Me.txtHash.Size = New System.Drawing.Size(247, 20)
        Me.txtHash.TabIndex = 4
        '
        'lblMD5
        '
        Me.lblMD5.AutoSize = True
        Me.lblMD5.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.lblMD5.Location = New System.Drawing.Point(9, 118)
        Me.lblMD5.Name = "lblMD5"
        Me.lblMD5.Size = New System.Drawing.Size(43, 16)
        Me.lblMD5.TabIndex = 5
        Me.lblMD5.Text = "MD5:"
        '
        'lblStatus
        '
        Me.lblStatus.AutoSize = True
        Me.lblStatus.Font = New System.Drawing.Font("Microsoft Sans Serif", 9.75!,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.lblStatus.Location = New System.Drawing.Point(9, 156)
        Me.lblStatus.Name = "lblStatus"
        Me.lblStatus.Size = New System.Drawing.Size(55, 16)
        Me.lblStatus.TabIndex = 6
        Me.lblStatus.Text = "Status:"
        '
        'lblResult
        '
        Me.lblResult.AutoSize = True
        Me.lblResult.Location = New System.Drawing.Point(70, 158)
        Me.lblResult.Name = "lblResult"
        Me.lblResult.Size = New System.Drawing.Size(0, 13)
        Me.lblResult.TabIndex = 7
        '
        'vitLogo
        '
        Me.vitLogo.Image = CType(resources.GetObject("vitLogo.Image"),
System.Drawing.Image)
        Me.vitLogo.Location = New System.Drawing.Point(28, 188)
        Me.vitLogo.Name = "vitLogo"
        Me.vitLogo.Size = New System.Drawing.Size(268, 85)
        Me.vitLogo.SizeMode = System.Windows.Forms.PictureBoxSizeMode.StretchImage
        Me.vitLogo.TabIndex = 8
        Me.vitLogo.TabStop = False
        '
        'OpenFileDialog1
        '
        Me.OpenFileDialog1.FileName = "OpenFileDialog1"
        '
        'lblFooter
        '
        Me.lblFooter.AutoSize = True
        Me.lblFooter.Font = New System.Drawing.Font("Segoe MDL2 Assets", 9.0!,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.lblFooter.Location = New System.Drawing.Point(37, 276)
        Me.lblFooter.Name = "lblFooter"
        Me.lblFooter.Size = New System.Drawing.Size(250, 12)
        Me.lblFooter.TabIndex = 10
        Me.lblFooter.Text = "Part of Final Year Project under the guidance of"
        '
        'Label1
        '
        Me.Label1.AutoSize = True
        Me.Label1.Font = New System.Drawing.Font("Segoe MDL2 Assets", 9.0!,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.Label1.Location = New System.Drawing.Point(105, 288)
        Me.Label1.Name = "Label1"
        Me.Label1.Size = New System.Drawing.Size(106, 12)
        Me.Label1.TabIndex = 11
        Me.Label1.Text = " Prof. Sivanesan S."
        '
        'Form1
        '
        Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 13.0!)
        Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
        Me.ClientSize = New System.Drawing.Size(322, 309)
        Me.Controls.Add(Me.Label1)
        Me.Controls.Add(Me.lblFooter)
        Me.Controls.Add(Me.vitLogo)
        Me.Controls.Add(Me.lblResult)
        Me.Controls.Add(Me.lblStatus)
        Me.Controls.Add(Me.lblMD5)
        Me.Controls.Add(Me.txtHash)
        Me.Controls.Add(Me.btnBrowse)
        Me.Controls.Add(Me.txtFilePath)
        Me.Controls.Add(Me.lblSig)
        Me.Controls.Add(Me.lblTitle)
        Me.Name = "Form1"
        Me.Text = "VIT Antivirus"
        CType(Me.vitLogo, System.ComponentModel.ISupportInitialize).EndInit()
        Me.ResumeLayout(False)
        Me.PerformLayout()

    End Sub

    Friend WithEvents lblTitle As Label
    Friend WithEvents lblSig As Label
    Friend WithEvents txtFilePath As TextBox
    Friend WithEvents btnBrowse As Button
    Friend WithEvents txtHash As TextBox
    Friend WithEvents lblMD5 As Label
    Friend WithEvents lblStatus As Label
    Friend WithEvents lblResult As Label
    Friend WithEvents vitLogo As PictureBox
    Friend WithEvents OpenFileDialog1 As OpenFileDialog
    Friend WithEvents lblFooter As Label
    Friend WithEvents Label1 As Label
```

```
End Class
```

Form1.vb

```
Imports System.IO
Imports System.Security
Imports System.Security.Cryptography
Public Class Form1

    Function md5_hash(ByVal file_name As String)
        Return hash_generator("md5", file_name)
    End Function

    Function hash_generator(ByRef hash_type As String, ByRef file_name As String)

        Dim hash
        hash = MD5.Create

        Dim hashValue() As Byte

        Dim filestream As FileStream = File.OpenRead(file_name)
        filestream.Position = 0
        hashValue = hash.ComputeHash(filestream)
        Dim hash_hex = PrintByteArray(hashValue)

        filestream.Close()

        Return hash_hex
    End Function

    Public Function PrintByteArray(ByRef array() As Byte)

        Dim hex_value As String = ""

        Dim i As Integer
        For i = 0 To array.Length - 1

            hex_value += array(i).ToString("x2")
        Next i

        Return hex_value.ToLower
    End Function

    Private Sub btnBrowse_Click(sender As Object, e As EventArgs) Handles
btnBrowse.Click
        If OpenFileDialog1.ShowDialog = Windows.Forms.DialogResult.OK Then
            Dim path As String = OpenFileDialog1.FileName
            txtFilePath.Text = path

            Dim sample As String
            sample = md5_hash(path)
            txtHash.Text = md5_hash(path)

            Using f As System.IO.FileStream = System.IO.File.OpenRead("md5.txt")
                Using s As System.IO.StreamReader = New System.IO.StreamReader(f)
                    While Not s.EndOfStream
                        Dim line As String = s.ReadLine

                        If (line = sample) Then
                            lblResult.Text = "Infected!"
```



```
                            lblResult.ForeColor = Color.Red
                        Else
                            lblResult.Text = "Clean!"
                            lblResult.ForeColor = Color.Green
                        End If
                    End While
                End Using
            End Using
        End If
    End Sub

    Private Sub Label1_Click(sender As Object, e As EventArgs) Handles
lblFooter.Click

    End Sub
End Class
```