Secure Chat CP300 Project

This project is developed in Java using the RSA Cryptosystem to encrypt and decrypt messages sent via a tcp/ip server.

Michael O Brien T00150213

Contents

[Project Description: 2](#_Toc361590977)

[Use Case Diagrams: 2](#_Toc361590978)

[RSA Algorithm Explanation: 2](#_Toc361590979)

[RSA Attacks: 2](#_Toc361590980)

[Factoring: 3](#_Toc361590981)

[Improper Usage: 3](#_Toc361590982)

[Side Channel Attack: 3](#_Toc361590983)

[RSA Algorithm Pseudo Code: 3](#_Toc361590984)

[Key Generation: 3](#_Toc361590985)

[Encryption 3](#_Toc361590986)

[Decryption 3](#_Toc361590987)

[Flow of Events: 3](#_Toc361590988)

[Class Diagrams: 4](#_Toc361590989)

[RSA Algorithm & RSA Math Classes 4](#_Toc361590990)

[Client Classes 5](#_Toc361590991)

[Server Classes 5](#_Toc361590992)

[RSA Algorithm Test Cases: 6](#_Toc361590993)

[Screen Shots: 6](#_Toc361590994)

[Conclusions: 6](#_Toc361590995)

[What was completed? 6](#_Toc361590996)

[What have I learned upon completing this project? 6](#_Toc361590997)

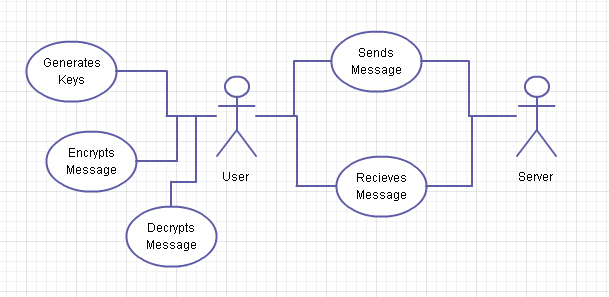
[What changes would I make to the project? 6](#_Toc361590998)

[Works Cited 7](#_Toc361590999)

# Project Description:

This application is designed so users can encrypt and decrypt messages using the RSA Algorithm sent via TCP/IP. This is to provide some level of privacy and security for those who want to send sensitive messages over a network. The user can randomly generate 2 keys and a modulo, which is a number that is specific to the two keys. The user shares his encryption key publically so that anyone can encrypt messages but only he can decrypt the message back again with his decryption key. There can be many users online at once and can chat freely and publically if they like without encrypted messages.

# Use Case Diagrams:



# RSA Algorithm Explanation:

In modern day cryptography it is based upon the theory of hard problems, the goal of cryptography is to show that breaking security systems is the same as solving some of the world’s greatest problems to date.

The foundations of cryptography are made on 3 crucial statements:

1. Cryptography is impossible without randomness.
2. Players are computationally limited by polynomial time algorithms.
3. Factoring is hard computationally. It is referred to as a “1-way trapdoor function”. It is easy to go from the factors to the product: (3011 \* 4793 = 14431723). It is difficult to do it in reverse, from the product to the factors: (14431723 =? \* ?) . This is due to the fact of the number having so many possible products that it takes a long time to check through each product to see if they are a match.

(Wayne & Sedgewick, 2011)

The RSA algorithm was invented by Ronald L. Rivest, Adi Shamir, and Leonard Adleman in 1977 and released into the public domain on September 6, 2000.

Public-key systems--or asymmetric cryptography--use two different keys with a mathematical relationship to each other. Their protection relies on the premise that knowing one key will not help you figure out the other. The RSA algorithm uses the fact that it’s easy to multiply two large prime numbers together and get a product. But you can’t take that product and reasonably guess the two original numbers, or guess one of the original primes if only the other is known. The public key and private keys are carefully generated using the RSA algorithm; they can be used to encrypt information or sign it.

(EMC)

## RSA Attacks:

### Factoring:

Factoring would be the most obvious way to break the cryptosystem

### Improper Usage:

### Side Channel Attack:

(Wayne & Sedgewick, 2011)

## How secure is the RSA’s n-bit key?

First of all the n-bit is the number of bits in the modulo,   
With current knowledge, "breaking" an RSA key by brute force effectively means factoring the modulus. The largest number that has been factored publically to date is RSA-640, a 640-bit number put up as a challenge by RSA and factored in 2005. This number took "only" around 350 CPU hours (using a cluster of 80 2.2 GHz Opterons). And the experimental software used by the team isn't exactly a "plug and play RSA cracker": it surely requires considerable configuration by somebody well versed in number theory.

Thorsten Kleinjung (one of the team that broke RSA-640) estimates that around 8.4 million CPU years are needed to factorise a 1024-bit number in software3 (his estimate is specifically 8.4 million uniprocessor PCs, taking into account memory and data transfer requirements). Using my favourite crude approximation, that's a million or so dollars of rented CPU time in 2009. It's not clear if and how this would scale to, say, several thousand 256-core machines (bearing in mind that that could be a fairly modest botnet by, say, 2020).

# RSA Algorithm Pseudo Code:

## Key Generation:

1. Choose two random prime numbers, P and Q.
2. Compute N (Modulo) = P \* Q.
3. Compute φ (Phi) = (p-1) \* (q-1).
4. Choose E such that 1 < E < φ and E and N are co-prime.
5. Compute a value for D such that (D \* E)

## Encryption

1. Encrypt

## Decryption

1. Decrypt

# Flow of Events:



# Class Diagrams:

## RSA Algorithm & RSA Math Classes



## Client Classes



## Server Classes



# RSA Algorithm Test Cases:

//Todo Develop Multiple Test Cases

Eg. Alice and Bob want to send a private message to each other

# Screen Shots:

//Screen shots of the algorithm working

# Conclusions:

## What was completed?

## What have I learned upon completing this project?

## What changes would I make to the project?

# Works Cited

EMC. (n.d.). *RSA Algorithm*. Retrieved July 14, 2013, from Information Security Glossary: http://www.rsa.com/glossary/?id=1045

Wayne, K., & Sedgewick, R. (2011, August 30). *Cryptography*. Retrieved July 09, 2013, from Introduction to Computer Science: http://introcs.cs.princeton.edu/java/78crypto/

# Other Sources

## Information on RSA Algorithm

<http://courses.cs.vt.edu/~cs5204/fall00/protection/rsa.html>  
<https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture12.pdf>  
<http://xtrmntr.org/priikone/docs/rsa.pdf> - Where I found that 65537 is a popular value for e and is if it is a multiple of phi to increment it by 2 till it is co-prime.