

EECE 455 - Cryptography and Network Security

**PUBLIC KEY CRYPTOGRAPHY USING RSA**

Project REPORT Done by:

*Alexander Arwadi* - 201702388

*Karl Chahine* - 201700591

*Cesar Bou Merhi* – 201702817

Supervised by:

Prof. *Ayman Tajeddine*

Date:

05/12/2018

Contents

[1- Introduction 3](#_Toc531254286)

[2- Background of RSA 3](#_Toc531254287)

[3- Our Work 3](#_Toc531254288)

[a) Tackling the problem 3](#_Toc531254289)

[b) The problems that we faced 4](#_Toc531254290)

[4- Testing 4](#_Toc531254291)

[5- How to run the code 5](#_Toc531254292)

[6- Group members efforts 6](#_Toc531254293)

[7- External References 6](#_Toc531254294)

# 1- Introduction

In this paper, we chose to implement the RSA algorithm using C++ code and package it in a graphical user interface (GUI) to make it user friendly and suitable for educational purposes. The RSA algorithm involves three main steps: key generation, encryption and decryption. These steps require knowledge in number theory which we were very interested in. Moreover, RSA is widely used today to send and receive secure data and as of today, RSA with keys of 1024 and 2048 bits are unbreakable. That is another reason why we chose RSA.

# 2- Background of RSA

RSA, short for Rivest–Shamir–Adleman, is an asymmetric public-key cryptosystem, widely used for secure data transmission. [6] It was first published in 1978, and later improved and developed. The main difficulty behind the Algorithm is the factorization of the product of two large prime numbers. In fact, in order to generate a key, one needs an integer n which is the product of two prime numbers p and q which are kept secret. Then the totient of n is calculated. The private key is chosen by the user such that it is relatively prime to the totient, and the public key is the multiplicative inverse of the private on with respect to the totient. Then the message/ciphertext is encrypted/decrypted using the following equations: C=Me and M=Cd mod n. The security of RSA is very high to the point that the 4096 bits key RSA is believed to be unbreakable in the foreseen future. [6]

# 3- Our Work

## a) Tackling the problem

We chose C++ to be our coding language. Therefore, we used Microsoft Visual Studio to do the project. In order to complete the project, we followed these steps:

* We learned how to create a GUI in Visual Studio via a YouTube tutorial [1]
* We searched for pre-made code online. However, we couldn’t find one which was implemented on a GUI. Therefore, we used parts of the pre-made code to begin with the project. [2]
* We had to understand how to convert the input from the GUI to a type readable by the compiler (converting from System::string to std::string and std::int). We used a code found online to convert from System::string to std::string. [3]
* We had to download a new library which can deal with very big numbers (1024 bits and 2048 bits). This library is called BigInteger. [4]
* We had to implement fast exponentiation to compute the powers efficiently. For that, another code found on the internet was used. [5]
* Also we got a modular inverse function online to compute modular inverses. [7]
* Finally, we had to combine all the above codes and steps together so that we have a homogeneous code that works perfectly. (and this step took forever)
* **Note: the code can be seen in Form1.h with all the necessary comments explaining every step and every loop. The codes are at the end of the file under the buttons’ sections.**

**The code in Form1.cpp is that of the GUI.**

Our program has the following features:

* Input p, q, e and message: calculates n, totient, d and ciphertext when Encrypt is clicked. If e and totient(n) are not relatively prime, the program will output “GCD(e, totient)!=1”.
* Input p, q, d and ciphertext: calculates n, totient, e and message when Decrypt is clicked.
* Input n, e and message: calculates ciphertext when Encrypt is clicked.
* Input n, d and ciphertext: calculates message when Decrypt is clicked.
* Generate all button generates everything for the user except for the message/ciphertext.
* The user can also import n, e and d. (How to do that is discussed in part 5).

## b) The problems that we faced

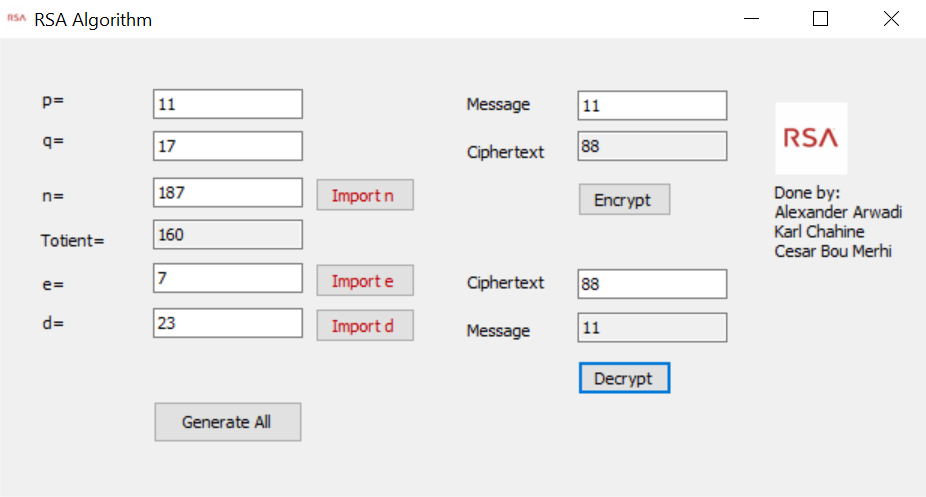
The main problem we faced during the project is how to make computations of very big numbers. We had to search a lot to find a library which handles this type of computations. Moreover, we had to understand how this library works.

Another problem faced was the conversion of the private key e from System::string to std::int. The process was not easy and we had to convert it first to std::string then to std::int.

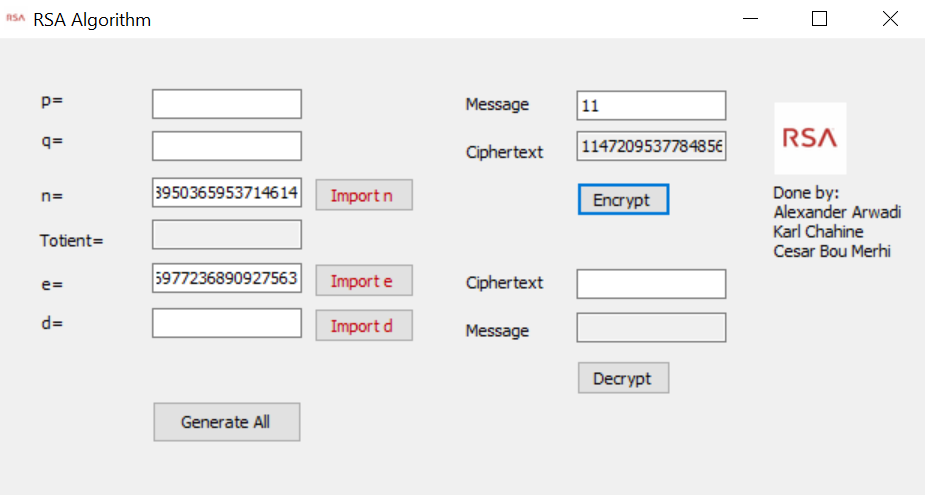
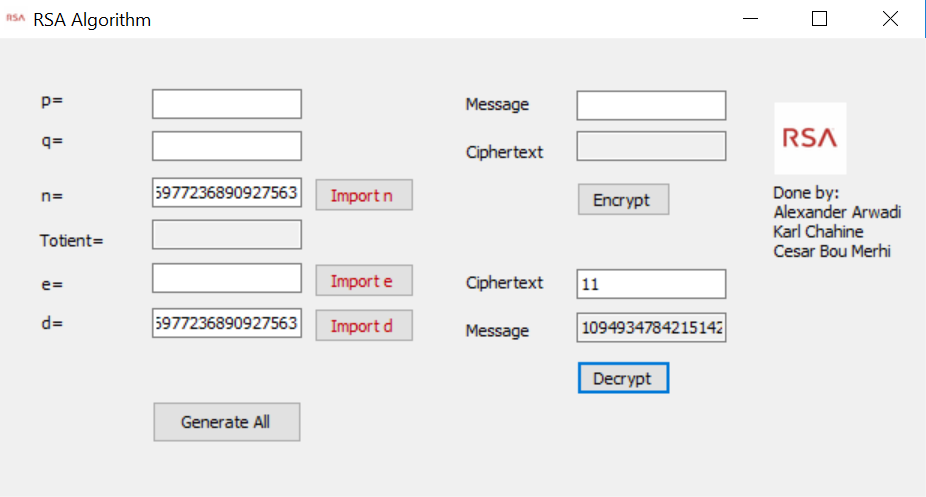
# 4- Testing

Several tests were made to make sure that our work is correct:

* We implemented the example provided in the slides to see if the basic encryption is working.
* We found the ciphertext of several messages then tried to decrypt the ciphertext and every time we obtained the same plaintext message.
* We found 1024 keys and 2048 keys online and tried to perform the first two tests again.
* To test the GCD function, we tried inputting a private key e which is not relatively prime to the totient.
* To test the Generate All and Import buttons, we just clicked them and saw the output.
* Screenshots:



Encryption and decryption using the example in the slides.



Encryption and decryption using 1024 bits key.

# 5- How to run the code

* Input p, q, e and message and click on Encrypt.
* Input p, q, d and ciphertext and click on Decrypt
* Input n, e and message and click on Encrypt.
* Input n, d and ciphertext and click on Decrypt.
* Generate all button generates everything for the user except for the message/ciphertext.
* The user can import n by putting the number in a file called **n.txt** and putting it in **C:** (i.e.: the directory is C:/n.txt).
* The user can import e by putting the number in a file called **e.txt** and putting it in **C:** (i.e.: the directory is C:/e.txt).
* The user can import d by putting the number in a file called **d.txt** and putting it in **C:** (i.e.: the directory is C:/d.txt).
* **Note that ciphertext/message should be less than n, e must be less than totient(n) AND all numbers are in decimal!**
* **Example of 1024 bits key:** 135066410865995223349603216278805969938881475605667027524485143851526510604859533833940287150571909441798207282164471551373680419703964191743046496589274256239341020864383202110372958725762358509643110564073501508187510676594629205563685529475213500852879416377328533906109750544334999811150056977236890927563
* **Example of 2048 bits key:** 25195908475657893494027183240048398571429282126204032027777137836043662020707595556264018525880784406918290641249515082189298559149176184502808489120072844992687392807287776735971418347270261896375014971824691165077613379859095700097330459748808428401797429100642458691817195118746121515172654632282216869987549182422433637259085141865462043576798423387184774447920739934236584823824281198163815010674810451660377306056201619676256133844143603833904414952634432190114657544454178424020924616515723350778707749817125772467962926386356373289912154831438167899885040445364023527381951378636564391212010397122822120720357

# 6- Group members efforts

* Alexander was responsible for making the codes of the Encrypt and Decrypt buttons. He also implemented the codes on the GUI.
* Karl was responsible for doing research and finding parts of the codes online. He also did the codes for import buttons.
* Cesar was responsible for the testing part. He also did the code for the Generate All button.
* We chose the program design together and the report was evenly distributed among us.

# 7- External References

[1] Hasan, M. (2016, October 09). Creating Simple GUI C program MS Visual Studio. Retrieved from <https://www.youtube.com/watch?v=QCnyqMWPkQk&t=610s>

[2] Mishra, N. (2017, March 15). RSA Algorithm in C and C (Encryption and Decryption). Retrieved from https://www.thecrazyprogrammer.com/2017/03/rsa-algorithm.html

[3] P. (2012, October 20). How to create a shared pointer to a managed string? Retrieved from https://stackoverflow.com/questions/12991402/how-to-create-a-shared-pointer-to-a-managed-string

[4] McCutchen, M. (2016, November 10). C Big Integer Library. Retrieved from https://mattmccutchen.net/bigint/

[5] Kumar, M. (2018, September 24). Modular Exponentiation (Power in Modular Arithmetic). Retrieved from <https://www.geeksforgeeks.org/modular-exponentiation-power-in-modular-arithmetic/>

[6] Robinson, Sara. “Still Guarding Secrets after Years of Attacks, RSA Earns Accolades for its Founders.” SIAM News, Volume 36, Number 5, June 2003.

[7] A. (2018, June 04). Modular multiplicative inverse. Retrieved from https://www.geeksforgeeks.org/multiplicative-inverse-under-modulo-m/