**IMPLEMENTATION OF RSA ALGORITHM**

By

Saurav Kumar 21ECB0A69

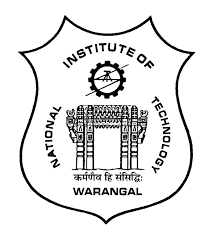
Yellanki Lakshmi Bhavani 21ECB0A68

Aditya Mishra21ECB0B03

Guided by:

**Dr. P. Prithvi Dr V. Narendar**

**Assistant Professor Assistant Professor**



**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

**NATIONAL INSTITUTE OF TECHNOLOGY, WARANGAL**

**2022-2023**

**CERTIFICATE**

This is to certify that the dissertation work entitled **INPLEMENTATION OF RSA ALGORITHM** is a bonafied record of work carried out work by Saurav Kumar (21ECB0A69) , Y Lakshmi Bhavani (21ECB0A69) and Aditya Mishra (21ECB0B03) submitted to faculty of “Electronics and Communication Engineering Department” , in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in “Electronics and Communication Engineering” at National Institute of Technology, Warangal during academic year (2022-2023).

**Dr. P. Prithvi Dr. V. Narendar**

Assistant Professor Assistant Professor

Department of Electronics and Department of Electronics and

Communication Engineering Communication Engineering

National Institute of Technology National Institute of Technology

Warangal Warangal

**ACKNOWLEDGEMENT**

I would like to express my deepest gratitude to my faculty in-charge **Dr. P. Prithvi,** **Assistant Professor**, Department of Electronics and Communication Engineering and **Dr. V. Narendar**, **Assistant Professor**, Department of Electronics and Communication Engineering, National Institute of Technology, Warangal for their constant supervision, guidance, suggestion and encouragement during this semester.

**ABSTRACT:**

RSA (Rivest–Shamir–Adleman) is a public-key cryptosystem that is widely used for secure data transmission. It is also one of the oldest. The acronym "RSA" comes from the surnames of **Ron Rivest, Adi Shamir and Leonard Adleman,** who publicly described the algorithm in 1977. An equivalent system was developed secretly in 1973 at Government Communications Headquarters (GCHQ) (the British signals intelligence agency) by the English mathematician Clifford Cocks. That system was declassified in 1997.

In a public-key cryptosystem, the encryption key is public and distinct from the decryption key, which is kept secret (private). An RSA user creates and publishes a public key based on two large prime numbers, along with an auxiliary value. The prime numbers are kept secret. Messages can be encrypted by anyone, via the public key, but can only be decoded by someone who knows the prime numbers.

The security of RSA relies on the practical difficulty of factoring the product of two large prime numbers, the "factoring problem". Breaking RSA encryption is known as the RSA problem. Whether it is as difficult as the factoring problem is an open question. There are no published methods to defeat the system if a large enough key is used.

RSA is a relatively slow algorithm. Because of this, it is not commonly used to directly encrypt user data. More often, RSA is used to transmit shared keys for symmetric-key cryptography, which are then used for bulk encryption–decryption.

**TABLE OF CONTENT:**

* Introduction
* Need for RSA algorithm
* Operation
  + - Key generation
    - Key distribution
    - Encryption
    - Decryption
* Flowchart
* Codes
  + - Main module
    - Key generating Module
    - D value module
    - Divider modules
    - Modular multiplication module
    - Test bench
* Simulations
  + - Waveform
    - Schematic
    - FPGA Implementation
* Conclusion

**INTRODUCTION:**

A patent describing the RSA algorithm was granted to MIT on 20 September 1983: U.S. Patent 4,405,829 "Cryptographic communications system and method". From DWPI's abstract of the patent:

The system includes a communications channel coupled to at least one terminal having an encoding device and to at least one terminal having a decoding device. A message-to-be-transferred is enciphered to cipher text at the encoding terminal by encoding the message as a number M in a predetermined set. That number is then raised to a first predetermined power (associated with the intended receiver) and finally computed. The remainder or residue, C, is... computed when the exponentiated number is divided by the product of two predetermined prime numbers (associated with the intended receiver).

A detailed description of the algorithm was published in August 1977, in Scientific American's Mathematical Games column. This preceded the patent's filing date of December 1977. Consequently, the patent had no legal standing outside the United States. Had Cocks’ work been publicly known, a patent in the United States would not have been legal either.

When the patent was issued, terms of patent were 17 years. The patent was about to expire on 21 September 2000**,** but RSA Security released the algorithm to the public domain on 6 September 2000.

**NEED FOR RSA ALGORITHM:**

RSA derives its security from the difficulty of factoring large integers that are the product of two large prime numbers. Multiplying these two numbers is easy, but determining the original prime numbers from the total -- or factoring -- is considered infeasible due to the time it would take using even today's supercomputers.

The public and private key generation algorithm is the most complex part of RSA cryptography. Two large prime numbers, p and q, are generated using the Rabin-Miller primality test algorithm. A modulus, n, is calculated by multiplying p and q. This number is used by both the public and private keys and provides the link between them. Its length, usually expressed in bits, is called the key length.

The public key consists of the modulus n and a public exponent, e, which is normally set at 65537, as it's a prime number that is not too large. The e figure doesn't have to be a secretly selected prime number, as the public key is shared with everyone.

The private key consists of the modulus n and the private exponent d, which is calculated using the Extended Euclidean algorithm to find the multiplicative inverse with respect to the totient of n.

**OPERATION:**

The RSA algorithm involves four steps: key generation, key distribution, encryption, and decryption.

A basic principle behind RSA is the observation that it is practical to find three very large positive integers e, d, and n, such that with modular exponentiation for all integers m (with 0 ≤ m < n):

(m^e)^d ≡ m (mod n)

and that knowing e and n, or even m, it can be extremely difficult to find d. The triple bar (≡) here denotes modular congruence (which is to say that when you divide (m^e)^d by n and m by n, they both have the same remainder).

In addition, for some operations it is convenient that the order of the two exponentiations can be changed and that this relation also implies

(m^d)^e ≡ m (mod n)

RSA involves a public key and a private key. The public key can be known by everyone and is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time by using the private key. The public key is represented by the integer n and e, and the private key by the integer d (although n is also used during the decryption process, so it might be considered to be a part of the private key too). “m” represents the message (previously prepared with a certain technique explained below).

**Key generation**

The keys for the RSA algorithm are generated in the following way:

1. Choose two large [prime numbers](https://en.wikipedia.org/wiki/Prime_number) *p* and *q*.
   * To make factoring harder, *p* and *q* should be chosen at random, be both large and have a large difference. For choosing them the standard method is to choose random integers and use a [primality test](https://en.wikipedia.org/wiki/Primality_test) until two primes are found.
   * *p* and *q* should be kept secret.
2. Compute *n* = *pq*.
   * *n* is used as the [modulus](https://en.wikipedia.org/wiki/Modular_arithmetic) for both the public and private keys. Its length, usually expressed in bits, is the [key length](https://en.wikipedia.org/wiki/Key_length).
   * *n* is released as part of the public key.
3. Compute *λ* (*n*), where *λ* is [Carmichael's totient function](https://en.wikipedia.org/wiki/Carmichael%27s_totient_function). Since *n* = *pq*, *λ*(*n*) = [lcm](https://en.wikipedia.org/wiki/Least_common_multiple)(*λ*(*p*), *λ*(*q*)), and since *p* and *q* are prime, *λ*(*p*) = [*φ*](https://en.wikipedia.org/wiki/Euler_totient_function)(*p*) = *p* − 1, and likewise *λ*(*q*) = *q* − 1. Hence *λ*(*n*) = lcm(*p* − 1, *q* − 1).
   * The lcm may be calculated through the [Euclidean algorithm](https://en.wikipedia.org/wiki/Euclidean_algorithm), since lcm (*a*, *b*) = |*ab*|/gcd(*a*, *b*).
   * *λ* (*n*) is kept secret.
4. Choose an integer *e* such that 2 < *e* < *λ* (*n*) and [gcd](https://en.wikipedia.org/wiki/Greatest_common_divisor)(*e*, *λ*(*n*)) = 1; that is, *e* and *λ*(*n*) are [coprime](https://en.wikipedia.org/wiki/Coprime).
   * *e* having a short [bit-length](https://en.wikipedia.org/wiki/Bit-length) and small [Hamming weight](https://en.wikipedia.org/wiki/Hamming_weight) results in more efficient encryption – the most commonly chosen value for *e* is 216 + 1 = 65537. The smallest (and fastest) possible value for *e* is 3, but such a small value for *e* has been shown to be less secure in some settings.[[15]](https://en.wikipedia.org/wiki/RSA_(cryptosystem)#cite_note-Boneh99-15)
   * *e* is released as part of the public key.
5. Determine *d* as *d* ≡ *e*−1 (mod *λ* (*n*)); that is, *d* is the [modular multiplicative inverse](https://en.wikipedia.org/wiki/Modular_multiplicative_inverse) of *e* modulo *λ*(*n*).
   * This means: solve for *d* the equation *de* ≡ 1 (mod *λ*(*n*)); *d* can be computed efficiently by using the [extended Euclidean algorithm](https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm), since, thanks to *e* and *λ*(*n*) being coprime, said equation is a form of [Bézout's identity](https://en.wikipedia.org/wiki/B%C3%A9zout%27s_identity), where *d* is one of the coefficients.
   * *d* is kept secret as the *private key exponent*.

The public key consists of the modulus n and the public (or encryption) exponent e. The private key consists of the private (or decryption) exponent d, which must be kept secret. p, q, and λ(n) must also be kept secret because they can be used to calculate d. In fact, they can all be discarded after d has been computed.

In the original RSA paper, the Euler totient function φ(n) = (p − 1)(q − 1) is used instead of λ(n) for calculating the private exponent d. Since φ(n) is always divisible by λ(n), the algorithm works as well. The possibility of using Euler totient function results also from Lagrange's theorem applied to the multiplicative group of integers modulo pq. Thus any d satisfying d⋅e ≡ 1 (mod φ(n)) also satisfies d⋅e ≡ 1 (mod λ(n)). However, computing d modulo φ(n) will sometimes yield a result that is larger than necessary (i.e. d > λ(n)). Most of the implementations of RSA will accept exponents generated using either method (if they use the private exponent d at all, rather than using the optimized decryption method based on the Chinese remainder theorem described below), but some standards such as FIPS 186-4 may require that d < λ(n). Any "oversized" private exponents not meeting this criterion may always be reduced modulo λ(n) to obtain a smaller equivalent exponent.

Since any common factors of (p − 1) and (q − 1) are present in the factorization of n − 1 = pq − 1 = (p − 1)(q − 1) + (p − 1) + (q − 1), it is recommended that (p − 1) and (q − 1) have only very small common factors, if any, besides the necessary 2.

Note: The authors of the original RSA paper carry out the key generation by choosing d and then computing e as the modular multiplicative inverse of d modulo φ(n), whereas most current implementations of RSA, such as those following PKCS#1, do the reverse (choose e and compute d). Since the chosen key can be small, whereas the computed key normally is not, the RSA paper's algorithm optimizes decryption compared to encryption, while the modern algorithm optimizes encryption instead.

**Key distribution**

Suppose that Bob wants to send information to Alice. If they decide to use RSA, Bob must know Alice's public key to encrypt the message, and Alice must use her private key to decrypt the message.

To enable Bob to send his encrypted messages, Alice transmits her public key (n, e) to Bob via a reliable, but not necessarily secret, route. Alice's private key (d) is never distributed.

**Encryption**

After Bob obtains Alice's public key, he can send a message M to Alice.

To do it, he first turns M (strictly speaking, the un-padded plaintext) into an integer m (strictly speaking, the padded plaintext), such that 0 ≤ m < n by using an agreed-upon reversible protocol known as a padding scheme. He then computes the cipher text c, using Alice's public key e, corresponding to

C ≡m^e(mod n)

This can be done reasonably quickly, even for very large numbers, using modular exponentiation. Bob then transmits c to Alice. Note that at least nine values of m will yield a cipher text c equal to m, but this is very unlikely to occur in practice.

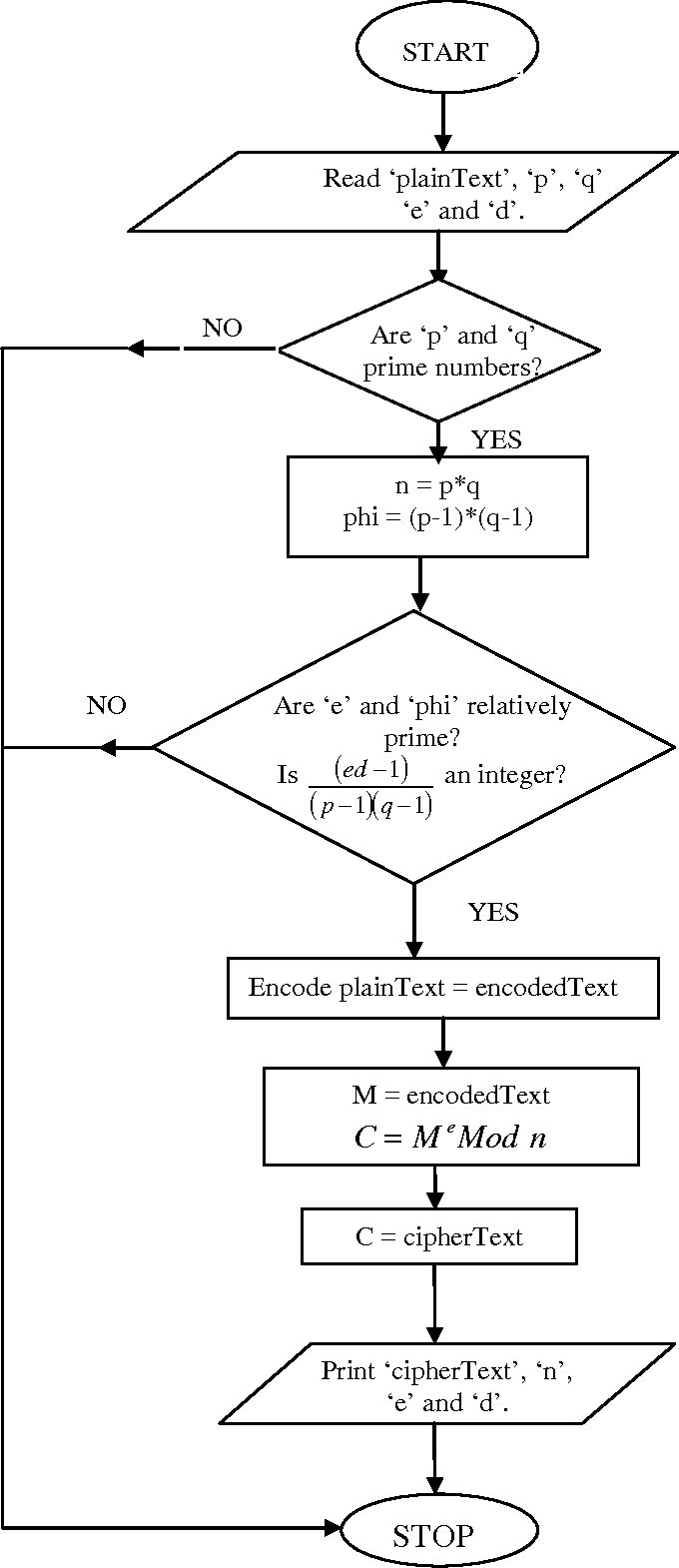
**Decryption**

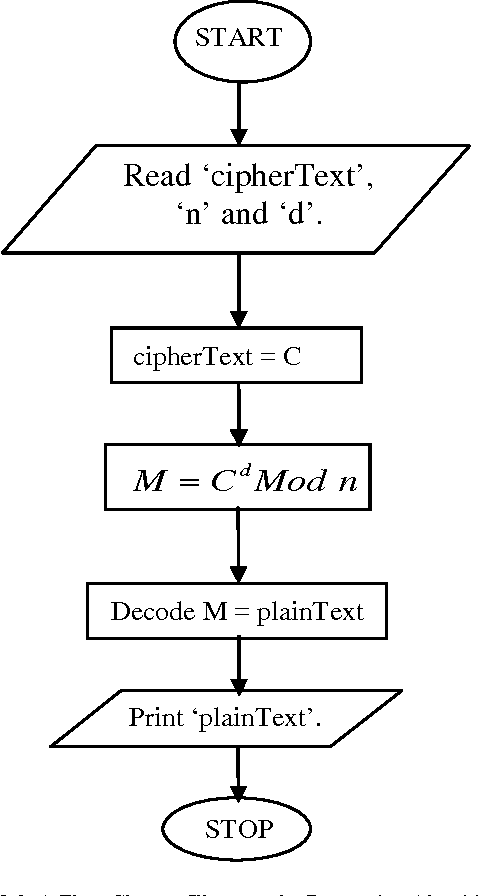
Alice can recover m from c by using her private key exponent d by computing

C^d ≡ (m^e)^d≡ m (mod n)

Given m, she can recover the original message M by reversing the padding scheme.

**FLOWCHART:**





**CODES (VERILOG HDL):**

**Main Module:**

*module Main ( input [3:0] M,input [1:0]p, input [1:0]q, input clk, input start,*

*input start1, input start2, output [1:0]e,output [3:0]n,output 3:0]remainder,*

*output [3:0] d,output finish,output fin1 );*

*keygen k1 (p,q,start,clk,e,finish);*

*dnew kd1 (p,q,e,clk,start1,n,d,fin1);*

*modularmult m1 (M,d,n,start2,clk,finished,Mpower,remainder);*

*endmodule*

**Key generation module:**

*module keygen (input [1:0]p,input [1:0]q,input start,input clk,output [1:0]e1,*

*output reg finish);*

*reg [3:0]e;*

*assign e1=e[1:0];*

*reg fin;*

*wire [3:0]phin;*

*assign phin=(p-1)\*(q-1);*

*reg [3:0]x,y,random,gcd;*

*wire [3:0]r,x1,y1;*

*Divider d2(x1,y1,outResult,r);*

*assign y1=y,x1=x;*

*always @(posedge clk)*

*begin*

*if(start)*

*begin*

*x<=phin;*

*random<=3;*

*y<=3;*

*gcd<=0;*

*fin<=0;*

*finish<=0;*

*e<=0;*

*end*

*if((fin==1) & (gcd==1))*

*begin*

*e<=random;*

*finish<=1;*

*end*

*if (r==0)*

*begin*

*gcd<=y;*

*fin<=1;*

*end*

*if( fin==0)*

*begin*

*x<=y;*

*y<=r;*

*end*

*if ((fin==1) & (gcd!=1)) begin*

*random<=random+2;*

*y<=random+2;*

*x<=phin;*

*gcd<=0;*

*fin<=0;*

*end*

*end*

*endmodule*

**Divider4 Module:**

*module Divider(A,B,Res,remainder);*

*parameter WIDTH = 4;*

*input [WIDTH-1:0] A;*

*input [WIDTH-1:0] B;*

*output [WIDTH-1:0] Res;*

*output reg [WIDTH-1:0] remainder;*

*reg [WIDTH-1:0] Res = 0;*

*reg [WIDTH-1:0] a1,b1;*

*reg [WIDTH:0] p1;*

*reg [1:0]remainderL;*

*integer i;*

*always@ (A or B)*

*begin*

*a1 = A;*

*b1 = B;*

*p1= 0;*

*for(i=0;i < WIDTH;i=i+1) begin*

*p1 = {p1[WIDTH-2:0],a1[WIDTH-1]};*

*a1[WIDTH-1:1] = a1[WIDTH-2:0];*

*p1 = p1-b1;*

*if(p1[WIDTH-1] == 1) begin*

*a1[0] = 0;*

*p1 = p1 + b1; end*

*else*

*a1[0] = 1;*

*end*

*Res = a1;*

*{remainderL,remainder} = p1;*

*end*

*endmodule*

**D value calculating Module:**

*module dnew (input [1:0] p, input [1:0] q, input [1:0] e1, input clk, input start,*

*output reg [3:0] n, output [3:0] d,output finished );*

*reg [11:0] A,B,C;*

*reg [3:0] G;*

*reg [3:0]e;*

*wire [3:0]outResult,Q;*

*Divider d2(A[3:0],B[3:0],outResult,remainder);*

*assign Q=outResult;*

*always@(posedge clk)*

*begin*

*if(start)*

*begin*

*e={2'b00,e1};*

*n=p\*q;*

*G=(p-1)\*(q-1);*

*A={4'h001,4'h000,G};*

*B={4'h000,4'h001,e};*

*end*

*else if(B[3:0]!=1)*

*begin*

*C[11:8]=A[11:8]-Q\*B[11:8];*

*C[7:4]=A[7:4]-Q\*B[7:4];*

*C[3:0]=A[3:0]-Q\*B[3:0];*

*A=B;*

*B=C;*

*end*

*end*

*assign d=B[7:4];*

*assign finished=B[3:0]==1;*

*endmodule*

**Modular Multiplication Module:**

*module modularmult (input [7:0]M,input [7:0]e,input [7:0]n,input start,*

*input clk,output finished,output reg[7:0]Mpower,output [3:0] remainder );*

*reg [7:0] ncount;*

*reg [15:0]x,n1;*

*Divider32 d1(x,n1,outResult,remainder);*

*always @(posedge clk)*

*begin*

*if(start) begin*

*ncount = e-1;*

*Mpower = M;*

*x=0;*

*n1={4'b0000,n};*

*end*

*else if(!finished)*

*begin*

*Mpower = remainder \* M;*

*ncount = ncount - 1;*

*end*

*x=Mpower;*

*end*

*assign finished = (ncount == 0)?1:0;*

*endmodule*

**Divider 8 Module:**

*module Divider32(A,B,Res,remainder);*

*parameter WIDTH = 8;*

*input [WIDTH-1:0] A;*

*input [WIDTH-1:0] B;*

*output [WIDTH-1:0] Res;*

*output reg [WIDTH-1:0] remainder;*

*reg [WIDTH-1:0] Res = 0;*

*reg [WIDTH-1:0] a1,b1;*

*reg [WIDTH:0] p1;*

*reg [1:0]remainderL;*

*integer i;*

*always@ (A or B)*

*begin*

*a1 = A;*

*b1 = B;*

*p1= 0;*

*for(i=0;i < WIDTH;i=i+1) begin*

*p1 = {p1[WIDTH-2:0],a1[WIDTH-1]};*

*a1[WIDTH-1:1] = a1[WIDTH-2:0];*

*p1 = p1-b1;*

*if(p1[WIDTH-1] == 1) begin*

*a1[0] = 0;*

*p1 = p1 + b1; end*

*else*

*a1[0] = 1;*

*end*

*Res = a1;*

*{remainderL,remainder} = p1;*

*end*

*endmodule*

**Test Bench:**

*module test\_bench;*

*reg [15:0] M;*

*reg [7:0] p;*

*reg [7:0] q;*

*reg clk;*

*reg start;*

*reg start1;*

*reg start2;*

*wire [7:0] e;*

*wire [15:0] n;*

*wire [15:0] remainder;*

*wire [15:0] d;*

*wire finish;*

*wire fin1;*

*Main uut (.M(M), .p(p), .q(q), .clk(clk),.start(start), .start1(start1), .start2(start2),*

*.e(e), .n(n), .remainder(remainder), .d(d), .finish(finish), .fin1(fin1));*

*initial begin*

*M = 0;p = 0;q = 0;clk = 0;*

*start = 0;start1 = 0;start2 = 0;#10;*

*M = 1256;*

*p = 67;q = 53;#10;*

*start = 1;#5;*

*start=0;#40;*

*start1=1;#10;*

*start1=0;#30;*

*start2=1; #10;start2=0;*

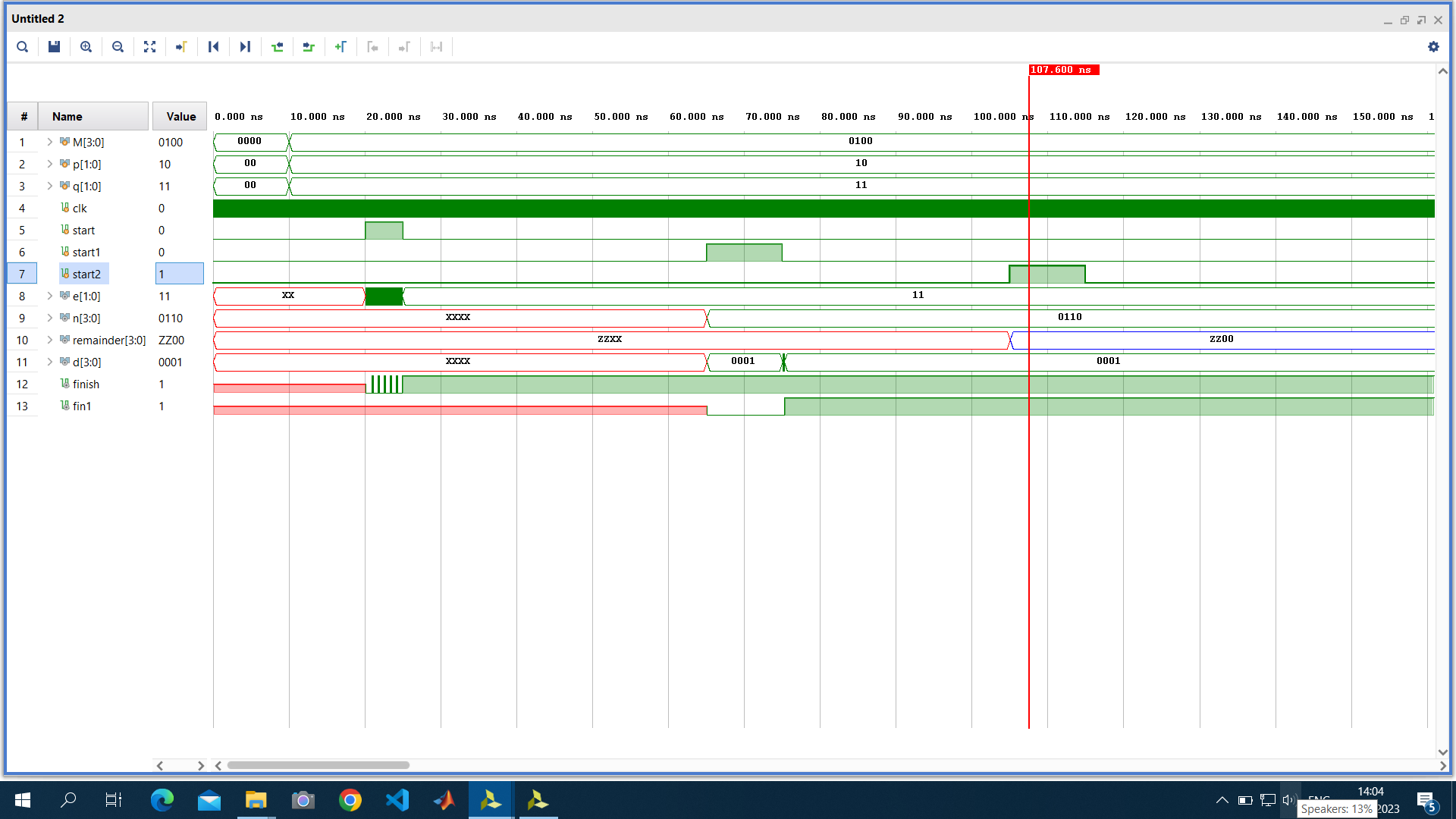
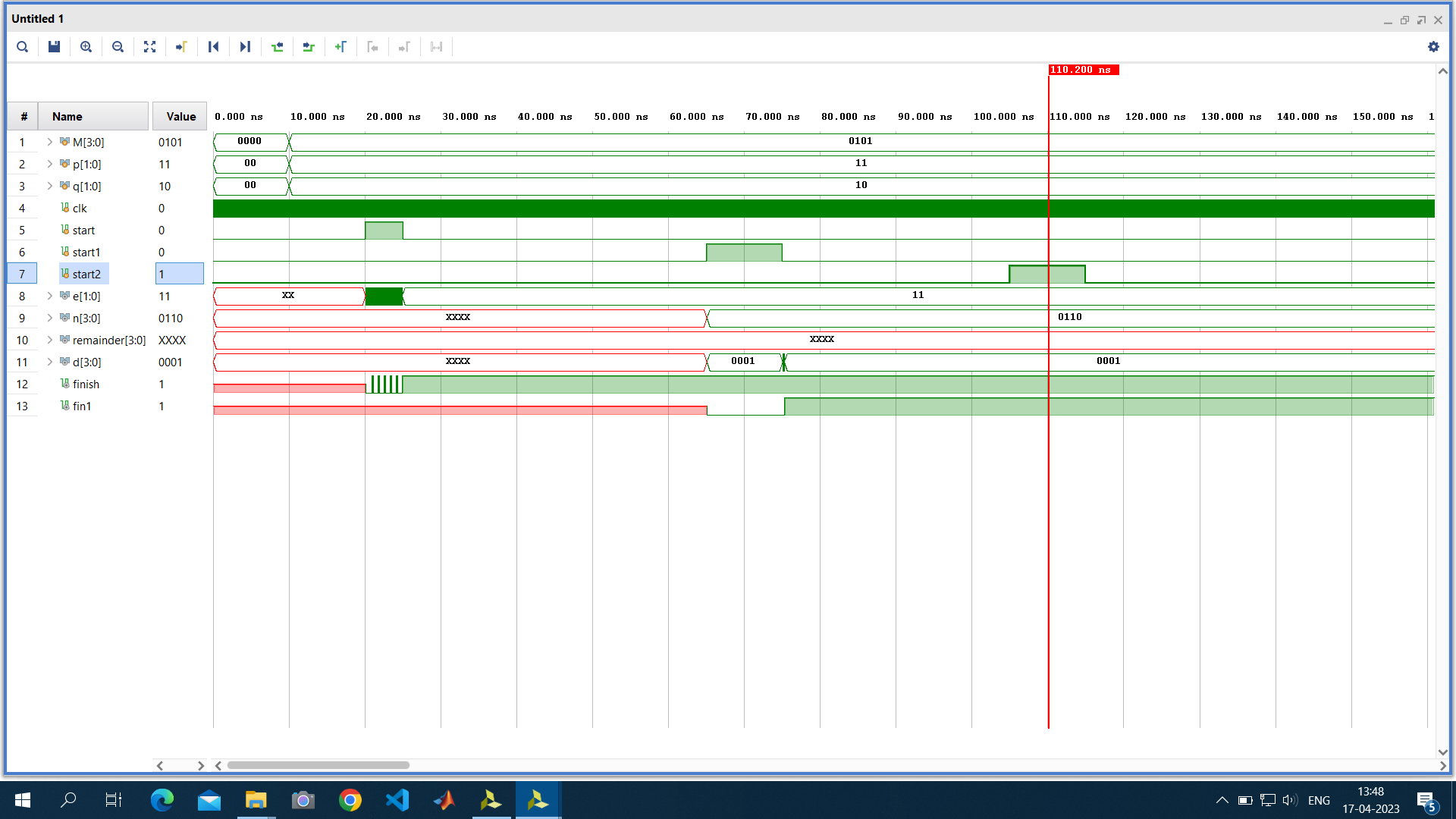
*end*

*always #0.1 clk=!clk;*

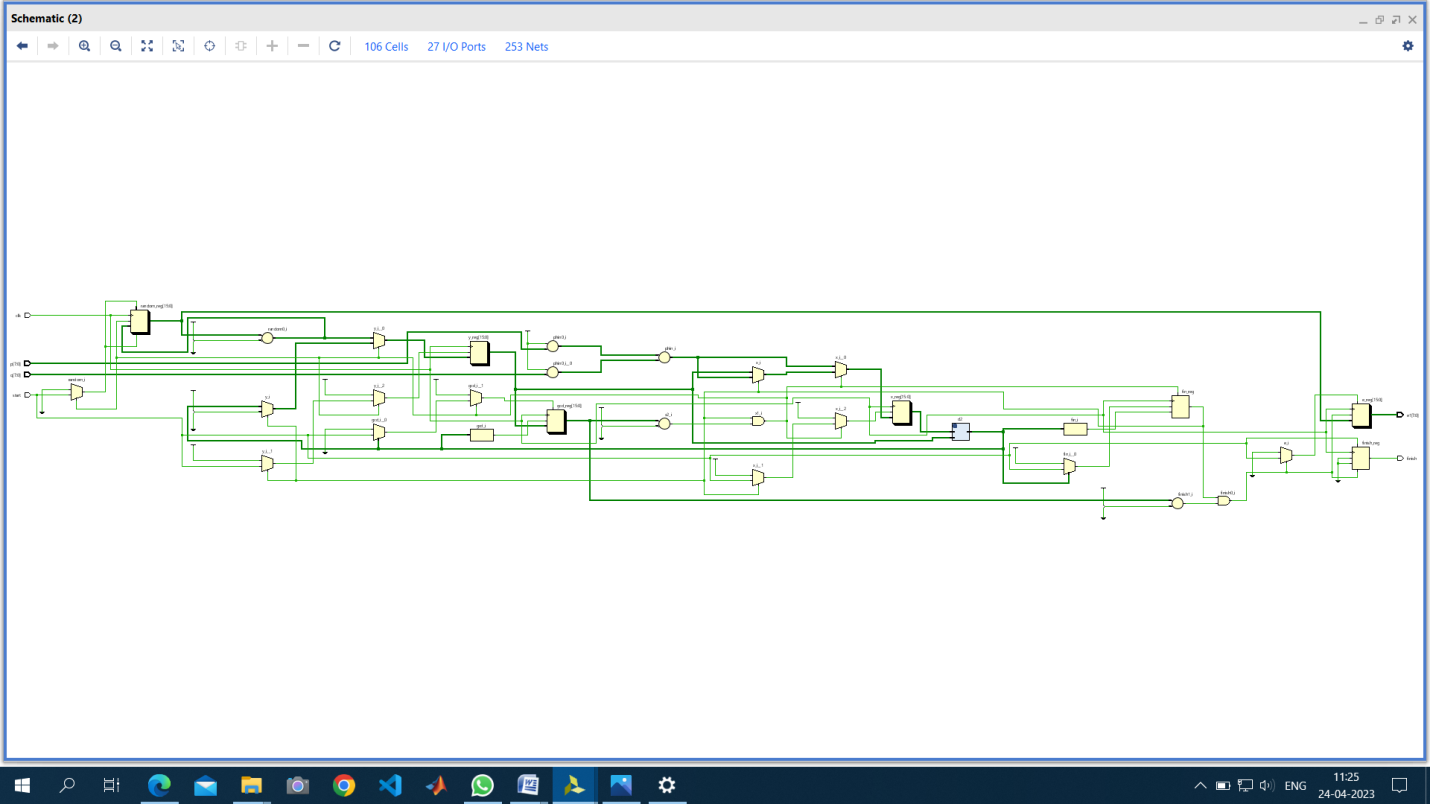
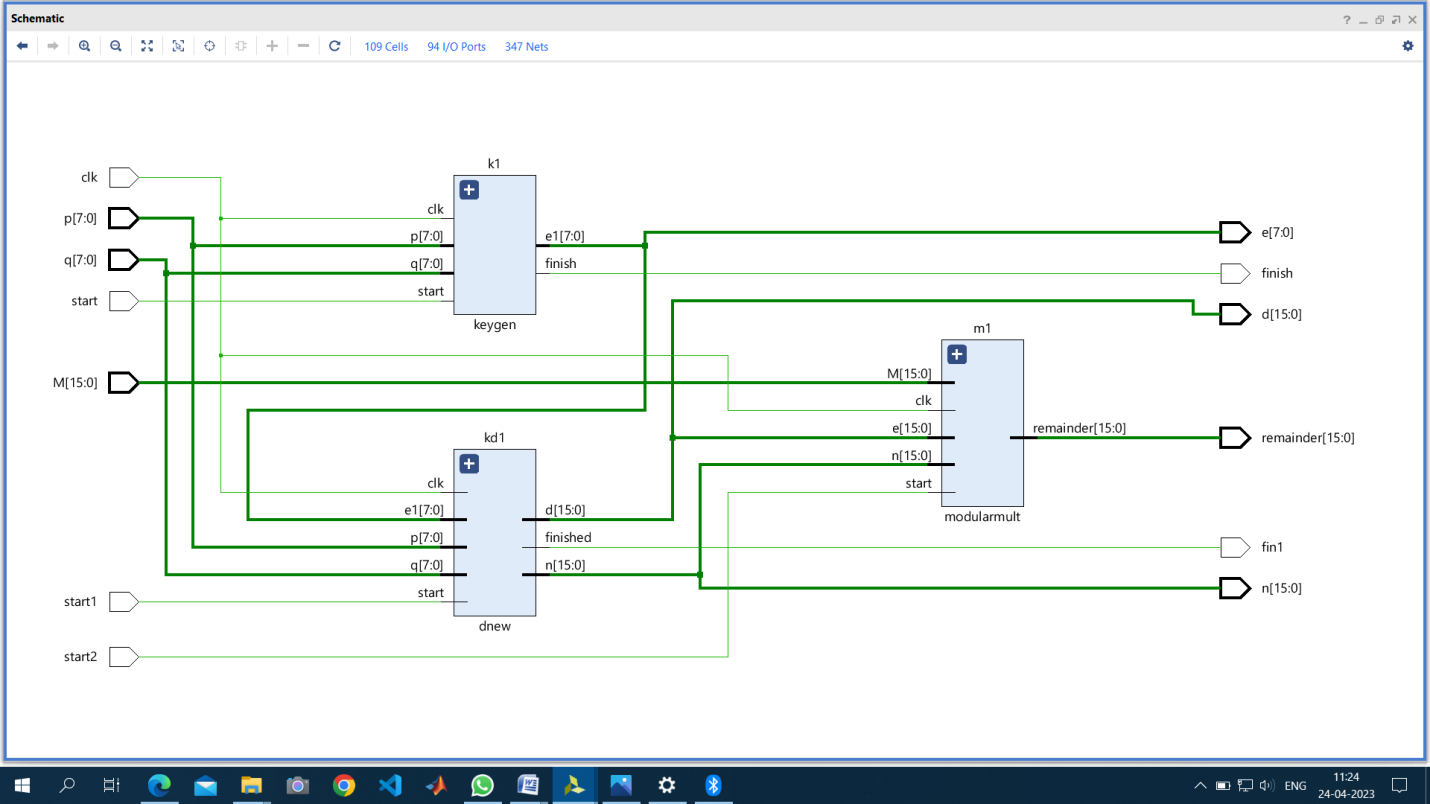
*endmodule*

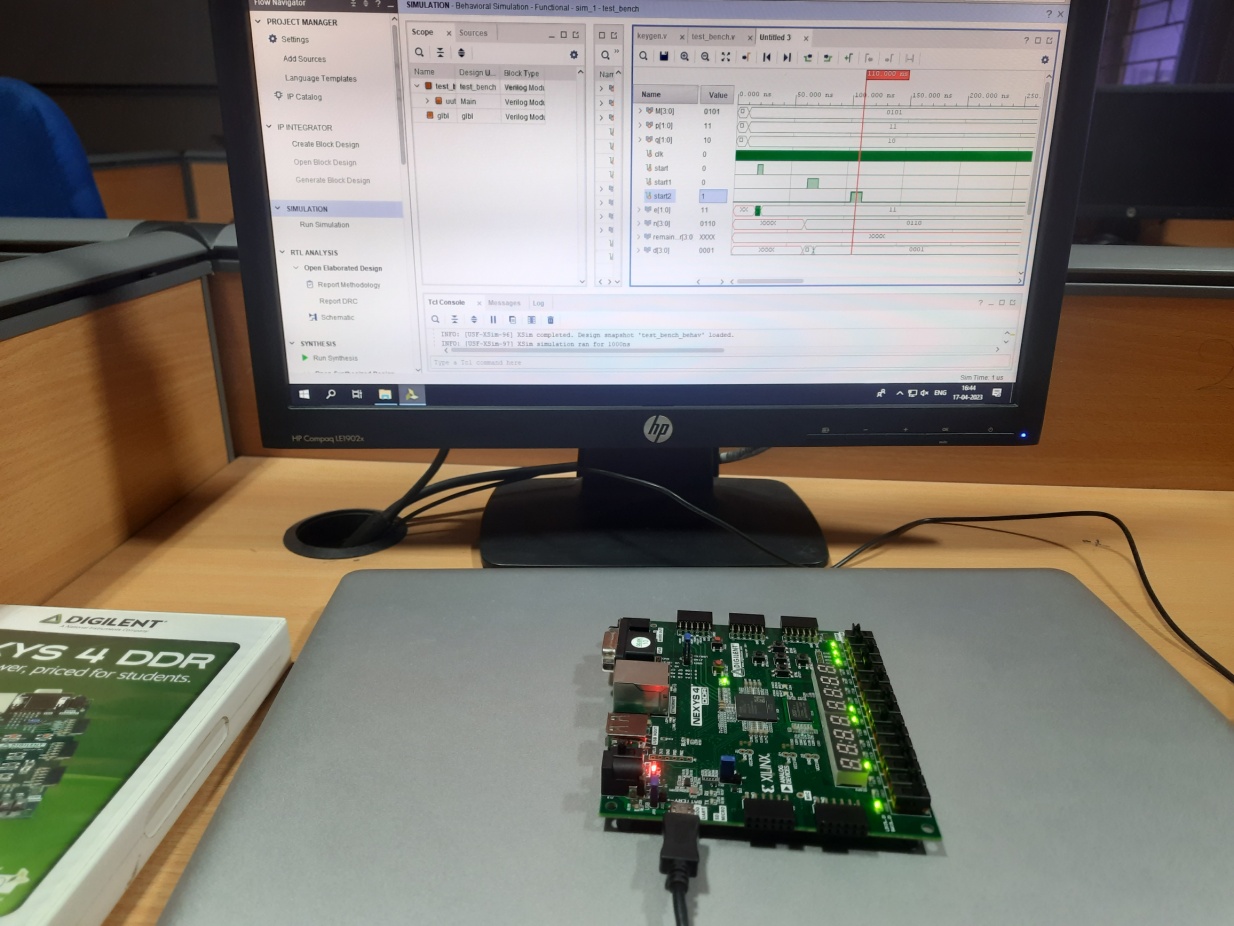
**SIMULATIONS:**

**Waveform-**



**RTL Schematic-**



**Nexys 4DDR Artix-7 (XC7A100TCSG324-1) FPGA Implementation-**

****

**CONCLUSION:**

We have discussed various prospects of RSA Algorithm in today’s scenario that gives the ability to user to accept encrypted data from various people who encrypt the data using their own public key and only the user can decrypt the data using their own private key.

There are some applications of RSA Algorithm are as follows −

* RSA algorithm is asymmetric cryptography algorithm as it operate on two different keys such as public key and private key. The public key is likely to everyone, and private key remains private. The public key includes two numbers, one of which is a multiplication of two large prime numbers.
* The RSA algorithm is based on the complexity included in the factorization of large numbers. The RSA algorithm depends on the fact that there is no effective method to factor very large numbers. Therefore, it can deducing an RSA key would take a large amount of time and processing power.
* In RSA encryption, a message is encrypted with a code is known as a public key, which does not required to be hidden. It is based on the mathematical features of the RSA algorithm, because a message has been encrypted with the public key, it can only be decrypted by another key, which is known as the private key. Therefore, a set of key, which are public and private keys, is needed to read such messages.
* The application of the RSA algorithm derives its security from factoring the huge integral component, which are the product of two large numbers. It is simply to multiple any of the figures.

The computation of the original primary numbers from the sum or variables is difficult because the time it takes even using supercomputers is the disadvantage of the RSA algorithm.

* The most ambiguous feature of RSA cryptography is the public and private key generation algorithm. They primarily test algorithm produced using the Rabin Miller test, which are p and q, the two large numbers.

A module, n, is calculated by multiplying p and q. This number can be used for a private and public key and supports the connection between them is known as the key length, and the length of the key is generally defined in bits.

* RSA encryption is generally used in combination with other encryption schemes, or for digital signatures which can validate the authenticity and integrity of a message. It cannot be used to encrypt entire messages or files, because it is less effective and more resource-heavy than symmetric-key encryption.
* In RSA public keys, two large, randomly produced prime factors contribute to their complexity. The numbers and sequences are generated randomly. RSA algorithm depends on using prime factorization as an approach of one-way encryption, so its complete security premise is based on its use.