

A Study of RSA Algorithm in Cryptography

Soe Moe Myint¹, Moe Moe Myint², Aye Aye Cho³

^{1,2}Lecturer, ³Associate Professor

¹Faculty of Computer Systems and Technologies, University of Computer Studies, Patheingyi, Myanmar

²Information Technology Support and Maintenance, University of Computer Studies, Patheingyi, Myanmar

³Faculty of Computer Science, University of Computer Studies, Hinthada, Myanmar

How to cite this paper: Soe Moe Myint | Moe Moe Myint | Aye Aye Cho "A Study of RSA Algorithm in Cryptography"

Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-3 | Issue-5, August 2019, pp.1670-1674, <https://doi.org/10.31142/ijtsrd26749>



IJTSRD26749

Copyright © 2019 by author(s) and International Journal of Trend in Scientific Research and Development Journal. This is an Open Access article distributed under the terms of the Creative Commons Attribution License (CC BY 4.0) (<http://creativecommons.org/licenses/by/4.0>)



ABSTRACT

RSA (Rivest–Shamir–Adleman) is an algorithm used by modern computers to encrypt and decrypt messages. The purpose of the paper is how to produce two different keys. This is also called public key cryptography, because one of the keys can be given to anyone. In this paper also represent how to separate unwanted character by using Linux command.

KEYWORDS: RSA algorithm, private key, public key

INTRODUCTION

Cryptography technique is one of the principal means to protect information security. Not only has it to ensure the information confidential, but also provides digital signature, authentication, secret sub-storage, system security and other functions. RSA is one of the first public-key cryptosystems and is widely used for secure data transmission.

BACKGROUND THEORY

RSA is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography, because one of the keys can be given to anyone. The other key must be kept private. The algorithm is based on the fact that finding the factors of a large composite number is difficult: when the integers are prime numbers, the problem is called prime factorization. It is also a key pair (public and private key) generator. [1]

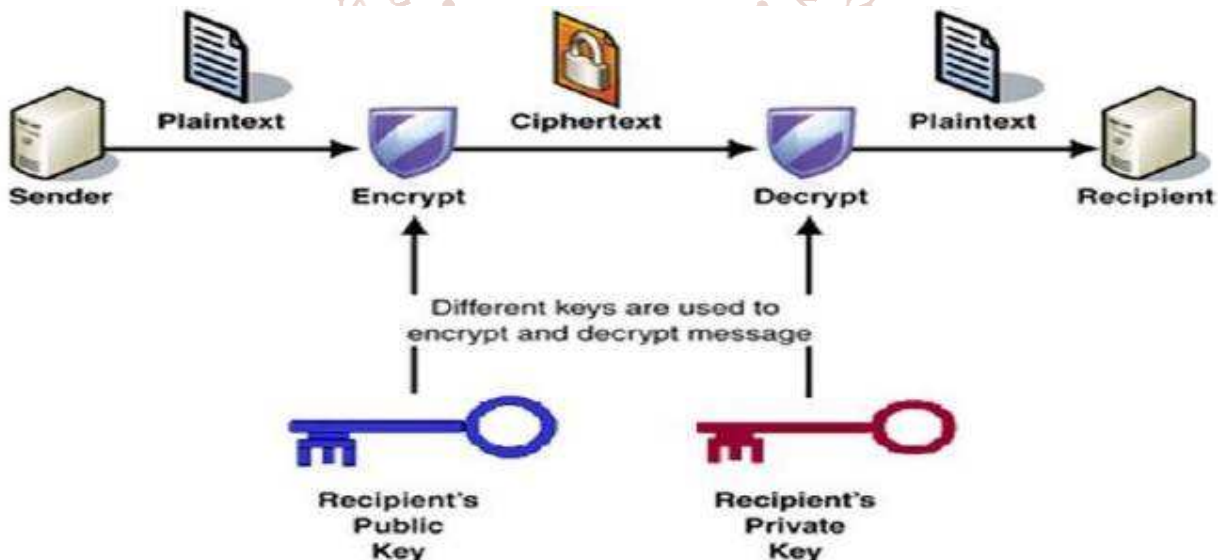


Fig 1 RSA algorithm

1. Getting Plain Text and Public Key

Step by step procedure by using public key

1.1 Using Openssl Command

Open Linux terminal under public.key folder and enter openssl command.

Command: openssl rsa -noout -text -inform PEM -in public.key

```

root@kali:~/Desktop/CT504/crypto100# openssl rsa -noout -text -inform PEM -in public.key -pubin
Public-Key: (2070 bit)
Modulus:
 25:b1:8b:f5:f3:89:09:7d:17:23:78:66:bb:51:cf:
 f8:de:92:24:53:74:9e:bc:40:3b:09:95:c9:7c:0e:
 38:6d:46:c1:61:ca:df:f7:7c:69:86:0d:ae:47:91:
 c2:14:cf:84:87:aa:aa:9f:26:e9:20:a9:77:83:49:
 06:03:8a:ef:b5:c3:08:27:df:cf:3f:c9:e9:76:95:
 44:f9:4e:07:cd:fe:08:72:03:9a:3a:62:62:11:66:
 78:b2:61:fb:2d:6b:9d:32:53:9e:92:a1:53:b3:67:
 56:29:ba:b3:94:2e:7d:35:e3:0f:7e:ef:5a:bf:1c:
 50:d7:97:d0:cc:88:e1:bd:cc:fd:1a:12:ea:6f:7e:
 f7:5c:37:27:db:df:2e:78:0f:34:28:ae:8f:7a:4f:
 b7:a8:9f:18:4a:36:50:32:b1:53:f8:42:5e:84:57:
 50:eb:2b:7a:bc:02:dc:15:ce:02:07:50:7a:a9:50:
 86:3b:b8:48:0a:78:02:8d:d6:29:79:94:4d:6c:63:
 3f:af:a1:03:e4:db:28:ce:87:f5:a0:c6:ed:4a:2f:
 26:64:42:7f:56:5c:77:81:ab:61:91:45:6d:97:1c:
 7f:fa:39:52:72:37:4c:ec:01:55:e5:f9:11:89:db:
 74:2e:4c:28:b0:3a:0f:a1:1c:ff:b0:31:73:d2:a4:
 cc:e6:ae:53
Exponent: 65537 (0x10001)

```

1.2 Removing unwanted characters

We get hexadecimal characters by using openssl command. But these characters are not real hexadecimal value. We remove unwanted characters and get the real hexadecimal value.

```

25b18bf5f389097d17237866bb51cf
f8de922453749ebc403b0995c97c0e
386d46c161cadff77c69860dae4791
c214cf8487aaaa9f26e920a9778349
06038aefb5c30827dfcf3fc9e97695
44f94e07cdfe0872039a3a62621166
78b261fb2d6b9d32539e92a153b367
5629bab3942e7d35e30f7eef5abf1c
50d797d0cc88e1bdccfd1a12ea6f7e
f75c3727dbdf2e780f3428ae8f7a4f
b7a89f184a365032b153f8425e8457
50eb2b7abc02dc15ce0207507aa950
863bb8480a78028dd62979944d6c63
3fafafa103e4db28ce87f5a0c6ed4a2f
2664427f565c7781ab6191456d971c
7ffa395272374cec0155e5f91189db
742e4c28b03a0fa11cfff03173d2a4
cce6ae53

```

1.3 Converting Hexadecimal to Decimal by using Python Program

```

def dec2hex(n):
    """return the hexadecimal string representation of integer n"""
    return "%X" % n

def hex2dec(s):
    """return the integer value of a hexadecimal string s"""
    return int(s, 16)

print "dec2hex(255) =", dec2hex(255) # FF
print "hex2dec('FF') =", hex2dec
('25b18bf5f389097d17237866bb51cf8de922453749ebc403b0995c97c0e386d46c161cadff77c69860dae4791c214cf8487aaaa9f26e920a977834906038aefb5c30827dfcf3fc9e97695
44f94e07cdfe0872039a3a6262116678b261fb2d6b9d32539e92a153b3675629bab3942e7d35e30f7eef5abf1c50d797d0cc88e1bdccfd1a12ea6f7ef75c3727dbdf2e780f3428ae8f7a4fb7a89f184a365032b153f8425e845750eb2b7abc02dc15ce0207507aa950863bb8480a78028dd62979944d6c633fafafa103e4db28ce87f5a0c6ed4a2f2664427f565c7781ab6191456d971c7ffa395272374cec0155e5f91189db742e4c28b03a0fa11cfff03173d2a4cce6ae53')

print "hex(255) =", hex(255) # 0xff
print "hex2dec('0xff') =", hex2dec('0xff') # 255

```

After running this python program, get decimal value

```

root@kali: ~/Desktop/backup/tool# python hexTodec.py
dec2hex(255) = FF
hex2dec('FF') = 7983218175733281855276461076134959298461474443227913532839899980
16278802836109003612812499731758050699162101795605064970751325249020868811203722
13626641879466491936860976686933630869673826972619938321951599146744807653301876
02657794957901833150277630398348556004648543103954170646714140826022009859276124
50106785923475018941762695805104597296336734680684671441997445637318263621026088
11033400887813754780282628099443490170016087838600998017490456601315802448567772
41162382628174724566095424541378151979429533619755968854353799219714225805322045
3757660537840276416475602759374950715283890232230741542737319569819793988431443
hex(255) = 0xff
hex2dec('0xff') = 255
    
```

1.3 Factorization by using Factor DB Website

After Factorize decimal value on Factor DB website, get p and q value.

The screenshot shows the FactorDB website interface. At the top, there are navigation tabs: Search, Sequences, Report results, Factor tables, Status, Downloads, and Login. A search bar contains the number 798321817573328185527646107613495929846147444322791353283989998016278802836109003612812499731758050691. A 'Factorize!' button is next to it. Below the search bar, a 'Result:' section shows the factorization: 7983218175...43 <623> = 3133337 * 2547832606...39 <617>. There are also links for 'More information' and 'ECM'. At the bottom, there is a footer with the text: 'factordb.com - 11 queries to generate this page (0.01 seconds) (Limits) (Imprint) (Privacy Policy) Just migrated to a new server. Leave me a message if problems occur. (markkutarroonen at web dot de)'

According to the result:

$$7983218175...43 = 3133337 \times 2547832606...39$$

Here value p is 3133337 and q is 2547832606...39.

1.4 Using RSA tool Python Program

Use RSA tool from python program and enter p and q value, then we get priv.pem file.

```

root@kali: ~/Desktop/1304/rsaTool-master# python rsatool.py p 3133337 q 254783260649374195922001721363994977199818429140262283154950962115931183219
1259936004725134841162947144246271486435695786036580117424411001355950996219648073788222782856382615209910833543894957393410121514115615648874284382
604006683085381436237908572039580231946285060290160568976167631915114735273080995755694004214429988739467674360776693782809447833640115044003587830688
371621654837427346238650838736771311207300401138341096789493055406758245324098102201192208337444273684804552967634136187123178716344146793367699088172
189217936816978728772470964266539999256625214484567800126283960890273067575342061776244939 -s priv.pem
Using (p, q) to init a RSA instance
n =
25c10bf5f389097d17237866bb51cf18de922453749ebc403b0995c97c0e366d46c161cadff77c69
860da44791c124cf8487aaaa9f26e528a977834996838ae7b5c30827d7cf3fc9e9769544f94e97cd
f6e8f2039a3e262116678b261fb2d5b9d32539a02a153b3675629cab3942a7d35e36f7eef5abf1c
50d797d8cc88e1bdccfd1a12ee6f7e7f5c3727dbaf2a78013420aef7a4fb7089f184a365032b153
78425e845755eb27b7abc82dc15ce0207507aa950883bc8486a78028ad629794446c633fafa03ea4
d28c8e97f5a6c6ed4a2f2964427f565c7781ab6191456d971c7ffa395272734cc0155a5f91189db
742e4c28b03a9fa11cfff83173d2a4c4e6ae53
e = 65537 (0x10001)
d =
312d72661f0c1b71469ca4c84c48af621dc5388f8f63591682b2accf3aee11b1932693ac46604c27
ae811740c54d70737056d9aab26dc830d3aee4277c7e083a91a70ed9bd40f0e370a060be18d2f4e
89635eadd5eb01d2ca87a5649c41a18165c37fd99a0f749f3c4a6c72c1fa5ed50e281de1de02278
5ab42572167e44b52a8c77f5dc54a2244928dc498aef30886bfc89029749c28f8adffae22b13c
30d1e440417195326778672bfc27a02c0b01af208de2d86386b7bfc7c0bfd240461f84d52a9e23
37b65b713fd55b0d29ed706f025143c17d1ae6aac5829e88959a752ae07a9948091c887d8e99a
d7f754bf7ae4968c898455de94ec63a8a721
p = 3133337 (0x2fcf99)
q =
30x3c4657875a7b1516941c23108b45f0b49bc62b83e4475f914048e6a52a787fc65fedd5ae864f82
012917bd9f469b6050ad7330b35bd2449f3fc4ada3c094f5f193bd7686d519eb7f93ac6c686f85
0f0aad27dc429695f99f0f0a4300292336e5e6ecf386b33f02e2fffb0371c245c6fa0a4d5cf8cc
75c019da375992009694847aa46ba0308085bf3d425b759a655f91699e214173fca5abd286ec8d
ca82bd29a7d3d51f41eb476ca87fd96bec28e22a08a5aac441ccea378f443e9dfbc0154a27e0d9
663fbc0a4bc1fe2cc6577cb79ff8b020fabb53c2579b007d10e112fc0732fbc4474d56d051ed5
    
```


Get private key file.

```

|-----BEGIN RSA PRIVATE KEY-----
MIIElQIBAAKCAQMsYv184kJfRcjEga7Uc/43pIkU3SevEA7CZXJfA44bUbBYc rf93xphg2uR5HC
FM+Eh6qqnybplKl3g0kGA4rvtcMIJ9/PP8npdpVE+U4Hzf4Icg0a0mJiEWZ4smH7LWudMl0ekqFT
s2dWkBgZlC59NeMpfu9avxxQ15fQzIjhvcz9GhLqB373XDcn298ueA80KK6Pek+3qJ8YSjZQMrfT
+EJehFdQ6yt6vALcFc4CB1B6qVCG07hICngCj dYpeZRNbGM/r6ED5Nsozof1oMbtSi8mZEJ/VLx3
gathkUVtlxx/+jLScjdM7AFV5fkRidT0LkwsDoPoRz/sDFz0qTM5q5TAgMBAECggECMS1yZh8M
G3FGnKTITEilsh3F0I+PY1kWgrKszzruEbGDNZ0sS2BMJ62DF0DFTXhzeFbQqrJtyDDT ruQnfH6I
OpGnigm9QPjuNwoGi++NL0q0LTXq3V6wHSyofVZAxB0YFlw3/ZCg90nzxKbPLB/l7VDigd4Q0CJ4
XbQlchZ+ZfTsqMd/XexU4iRjKA20m0jzAIA/yJkpdJzCj4rd/iKxDDDR70CEF/hT0md4Zyv8J6gs
iwGvIG3i2G0Gt7/HwL/SQEYfhNkqniM3tltxP9tVu9Ke19bwJRQ8F9GuauXI0CNaadi7vB6yZQJ
4cCH20lu1/dUv3rkl0yZhfXel0xjppq8hAgMvz5kCggEBAMnTxKV49ue/YwLBwjEATf/bSbyysD5E
dfkUBAbLknH/xl/t1a6GTWIBKRe9n0abYFCNczCzW2JEjz/EraPaLPX/Cb3XaG1Rm7f50sbGho+F
jwqtsn3EKwL fCP34pDACKjNu5ebs845rM/AuL/uDccJFvxvEpFz47MdsAZ2j9ZliAGiUhhRUA9A4
uFv8PUJbdZq1XwFpmyFBc/ymq9KG7G3Kgr1ian09UfQetHb0V/2Wvssg4joIppq7MThz0N49EPp37
wBVKJ+vQtj++/OS84f4uxld3y3j/iwIP67Y8JXmwB9FuES/Acy+8RH1FbUUE1ZNfQaxqjNouXTRd
ZYJPKMsCAwx6sQKCAQB5XE2y8roFQJ9im5gZv0K3ITwFsi0RCJsVAzX2JVhP/QZwvSp5B6tBfx
nqRX4LZZubS6ZB9fR7qbrbh77yGjimhhL1Yr5has2cDuJhJj2vvYf/oEhiAgrHTLwud3txQSuWyl
H3aU/QG00ze/FZsiJrMvQ/tRrJ00jU2rbRwRz0xP1n7THUh3PKQfK93q0PT0wqE0SGJv7NvB4LcR
MPCaVfUpZbSC+ox9Lrl1dz6Rzk0MAYoH04x/L3sI9zeRfofoL6k5JA49TpNIYZ/QK4P5REcf8Xj4
mTENXGVwf1pJggAx fu32uNKKsbq9WTILji7/Hxhuh00Nj r0c+UxAv3dhAgMuK/k=
-----END RSA PRIVATE KEY-----
    
```

2. Reading Flag (Encrypted File)

2.1 Open flag.enc file, see base64 format characters that are not real base64 format characters.

```

CQGd9sC/h9lnLpua50/071knSsP4N8wmdRsj0NIdfclrBhMjp7NoM5xy2S1NLLC2
yh7wbRw08nwj06UF4tmGKKfcjPcb4l4bFa5uvyMY1nJBvmqYlDbiCns0DjhpB1B
JfdpU1LUKtwsCxbc7fPL/zzUdwg0+of/R9wmM+Q0BPagTANbJo0mpDYxvNKRjvac
9Bw4CQTTh87moqsNRSE/Ik5tV2pkFRZfQxAZWuVePsHp0RXVithwvKzwmN9vMqGm
57Wb2Sto64db4gLDJh9GR0QN+EQh3yLoSS8NntBrZCDddzfkHa8wv6zN/5zvnBst
sDBkGyi88NzQxw9k0GjCwtwPw==
    
```

2.2 Remove unwanted characters

Remove unwanted characters from flag.enc file by using sed command.

Commands: sed -e ':a;N;\$!ba;s/ //g;s/\n//g' flag.enc

```

root@kali:~/Desktop/CT504/rsatool-master# sed -e ':a;N;$!ba;s/ //g;s/\n//g' flag.enc
CQGd9sC/h9lnLpua50/071knSsP4N8wmdRsj0NIdfclrBhMjp7NoM5xy2S1NLLC2yh7wbRw08nwj06UF4tmGKKfcjPcb4l4bFa5uvyMY1nJBvmqYlDbiCns0DjhpB1BJfdpU1LUKtwsCxbc7fPL/zzUdwg0+of/R9wmM+Q0BPagTANbJo0mpDYxvNKRjvac9Bw4CQTTh87moqsNRSE/Ik5tV2pkFRZfQxAZWuVePsHp0RXVithwvKzwmN9vMqGm57Wb2Sto64db4gLDJh9GR0QN+EQh3yLoSS8NntBrZCDddzfkHa8wv6zN/5zvnBstsDBkGyi88NzQxw9k0GjCwtwPw==root@kali:~/Desktop/CT504/rsatool-master#
    
```

2.3 Decryption with python program

Use python program to decrypt base64 characters with private key file.

```

def decrypt_RSA(private_key_loc, package):
    from Crypto.PublicKey import RSA
    from Crypto.Cipher import PKCS1_OAEP
    from base64 import b64decode
    key = open(private_key_loc, "r").read()
    rsakey = RSA.importKey(key)
    rsaobj = PKCS1_OAEP.new(rsakey)
    decrypted = rsaobj.decrypt(b64decode(package))
    return decrypted

flag = 'CQGd9sC/h9lnLpua50/071knSsP4N8wmdRsj0NIdfclrBhMjp7NoM5xy2S1NLLC2yh7wbRw08nwj06UF4tmGKKfcjPcb4l4bFa5uvyMY1nJBvmqYlDbiCns0DjhpB1BJfdpU1LUKtwsCxbc7fPL/zzUdwg0+of/R9wmM+Q0BPagTANbJo0mpDYxvNKRjvac9Bw4CQTTh87moqsNRSE/Ik5tV2pkFRZfQxAZWuVePsHp0RXVithwvKzwmN9vMqGm57Wb2Sto64db4gLDJh9GR0QN+EQh3yLoSS8NntBrZCDddzfkHa8wv6zN/5zvnBstsDBkGyi88NzQxw9k0GjCwtwPw=='
print decrypt_RSA('private.pem', flag)
    
```

After running Python program, we get the plain text.

EKO {classic_rsa_challenge_is_boring_but_necessary}.

```
root@kali:~/Desktop/CT504/rsatool-master# python decrypt.py
EKO{classic_rsa_challenge_is_boring_but_necessary}
```

Conclusion

Today, public key encryption and is widely used to secure sensitive data, particularly when it is being sent over in secure network such as the internet. Therefore, this paper describes how to work key generation in cryptography. RSA key is a private key based on RSA algorithm. Encryption is used for a secure symmetric key exchange that is used for actual transmitted data encryption .

References

- [1] https://simple.wikipedia.org/wiki/RSA_algorithm.
- [2] Behrouz A. Forouzan, Cryptography and Network

Security, McGraw-Hill International edition, 2008.

- [3] Cryptography & Network Security (project_paper) , University of Computer Studies, Patheingyi, Myanmar,2018
- [4] <http://mathworld.wolfram.com/RSAEncryption.html>
- [5] https://www.tutorialspoint.com/cryptography/public_key_encryption.
- [6] <https://ieeexplore.ieee.org/document/6021216>
- [7] https://www.schneier.com/blog/archives/2005/08/new_cryptanalyt.html [Accessed : Oct. 7, 2014]

