MADE BY - Shikher Srivastava (https://github.com/shikher7/)

Abstract

As technology advances there is an increase in the amount of data being transferred over the network. One of the principle challenge that is faced is security. Security is achieved by encryption; In this project, a AES symmetric block cipher is initially used to encrypt the file, the key obtained from the AES encryption is encrypted again using bit RSA algorithm. Similarly, the decryption is done using RSA algorithm to obtain the key which is used to decrypt the message using AES algorithm. AES is a symmetric algorithm that uses only a private key and RSA is an asymmetric encryption system that works with two different keys: A public and a private key. Both work complementary to each other, which means, when a message is encrypted with one of them can only be decrypted by its counterpart. This combination of algorithms provides better security and efficiency. The file served on a Tor server from which anyone can download by accessing the onion link but only person with private key will be able to decrypt the file.

Literature Review

Komal Rege et.al proposed a hybrid encryption scheme for Bluetooth communication security, using AES and RSA. The key of 128-bit is encrypted using RSA algorithm, similarly the message of sender is encrypted using AES cipher. Both encrypted AES-key and cipher text of message is used to generate a complex message, which is transmitted over the network. The decryption is exactly the reverse process of encryption algorithm. Palanisamy et.al proposes a hybrid cryptography technique using RSA and AES algorithms. RSA algorithm uses a key size of 128-bytes. It uses two pairs of keys: public key and private key. One pair is used at sender site for encryption/decryption and the other one at receiver's site. Ali E. Taki El_Deen [3] has proposed a hybrid encryption algorithm using AES and Blowfish. Plaintext of 64-bit is encrypted using Blowfish algorithm generating 64-bit cipher text which is again encrypted using same algorithm, thus generating a new 64-bit cipher text. The two outputs 64+64=128-bit cipher text is now given as an input to AES algorithm, generating the final 128-bit cipher text. Blowfish and AES algorithms make use of 32-bit and 128-bit key size respectively for their rounds. A statistical comparison of AES, DES, RSA, and Blowfish algorithms has been also provided. Ritu Pahal et.al [6] proposes an efficient implementation of AES. Instead of conventional 128-bit input, 200-bit input is copied into an array of 5*5 matrixes. The first nine rounds are same consisting of four transformations: Substitute Bytes, Shift Rows, Mix Columns, and Add Round Key, but in the final (10th) round transformation Mix Columns is not used. The results of proposed scheme are compared with 128-bit, 192-bit, and 256-bit AES techniques at the end.

Objective of the project

Sharing Files popularity has grown up rapidly as people want to share files in more secure and fast way possible. Today file sharing is used in almost all digital sector. Sharing files on this large scale raise questions on its security because it introduces risks of hacking, malware infection and loss of sensitive information. Companies share sensitive data and if there is chance of security breach than importance of sharing file is out weighted. That’s why we have done work on sharing file more securely and anonymously using tor services and used both AES and RSA to encrypt the file and the key respectively. The combination of the Tor service and dual encryption allows us to provide a highly file secure sharing medium.

REQUIREMENTS

Software requirements

Software Requirement:

- Python 3.5
- Tor Browser
- Tor Service
- Python IDE

Module used are as follows:

Stem

Stem is a Python controller library for Tor. With it we can use Tor's control protocol to script against the Tor process, or build things such as Nyx. Stem's latest version is **1.6.0**.

PyQt5

Qt is a set of C++ libraries and development tools that includes platform independent abstractions for graphical user interfaces, networking, threads, regular expressions, SQL databases, SVG, OpenGL, XML, user and application settings, positioning and location services, short range communications (NFC and Bluetooth) and access to the cloud. PyQt5 implements over 1000 of these classes as a set of Python modules.

Pycrypto

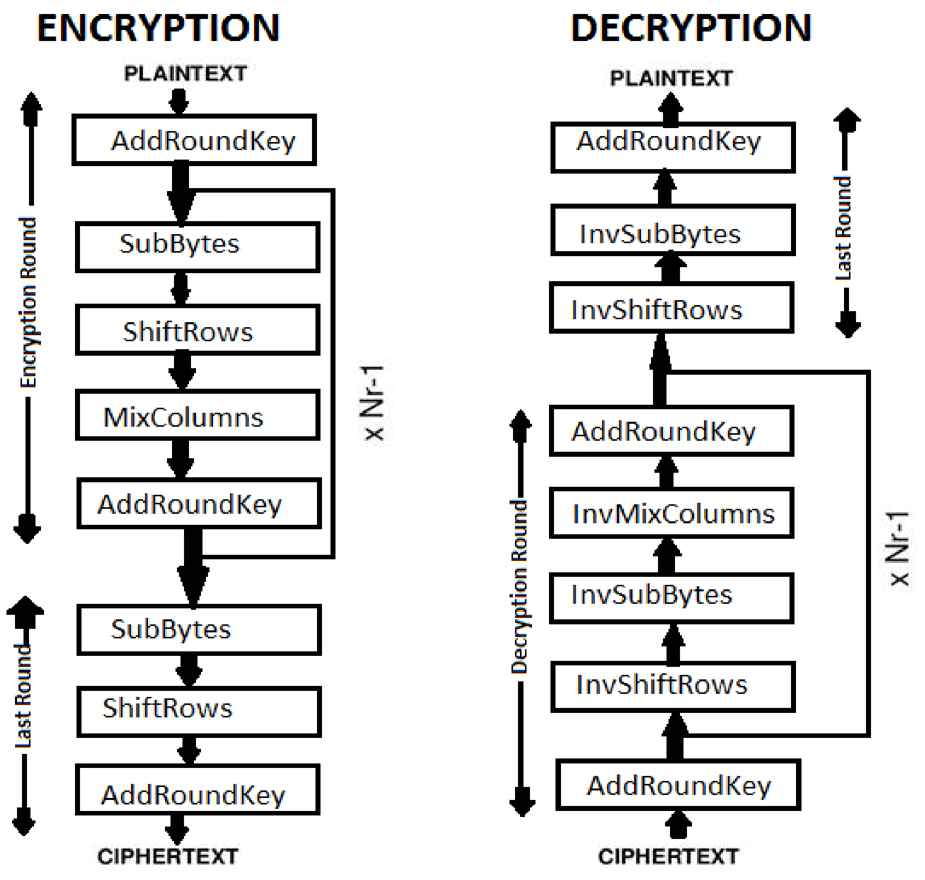
This is a collection of both secure hash functions (such as SHA256 and RIPEMD160), and various encryption algorithms (AES, DES, RSA, ElGamal, etc.). The package is structured to make adding new modules easy.

METHODOLOGIES USED

We have used two types of encryption and decryption technique they are

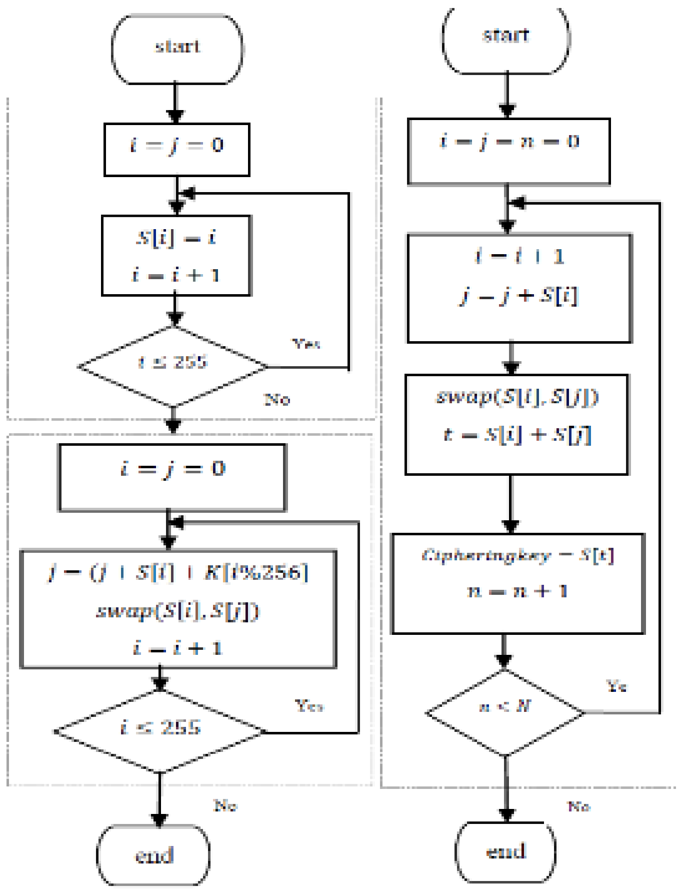
AES(Advanced Encryption Standard)

AES is one of the most secure type of encryption and decryption technique developed by U.S government to secure sensitive information and is used by most of software and hardware product developing industries. The AES encryption algorithm defines various changes that are to be performed on information put away in a cluster. The initial step of the figure is to put the information into a cluster; after which the figure changes are rehashed over various encryption rounds. The number of rounds is known by the key length, with 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys.



RSA(Rivest–Shamir–Adleman)

RSA is a relatively slow algorithm, and because of this, it is less commonly used to directly encrypt user data. More often, RSA passes encrypted shared keys for symmetric key cryptography which in turn can perform bulk encryption-decryption operations at much higher speed. The RSA algorithm involves four steps: key generation, key distribution, encryption and decryption. RSA involves a public key and a private key. The public key can be known by everyone, and it is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a less time by using the private key.



IMPLEMENTATION

We have used these two techniques in this file sharing project and also used tor services for securely sharing files.

Step 1: Browse the file to be encrypted.

Step 2: Now will apply AES encryption on browsed file using randomly generated key.

Step 3: From AES encryption will have key and encrypted file.

Step 4: Key generated from AES will be encrypted using RSA and it will use public key of receiver to encrypt given data (key from AES).

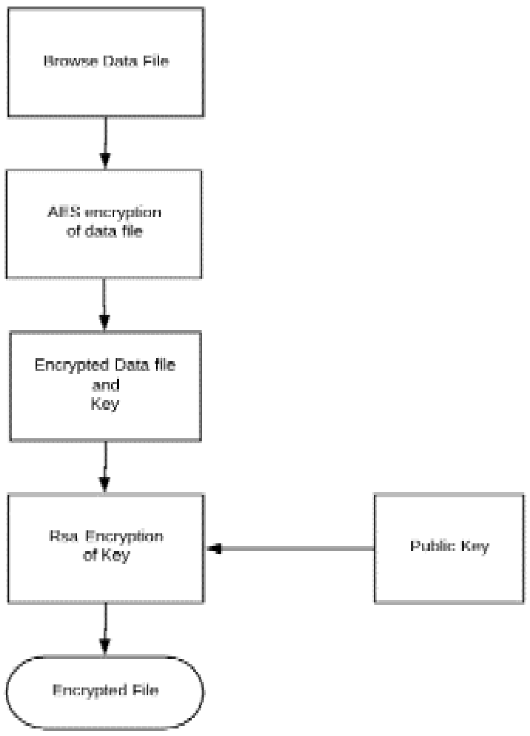
Step 5: Now we have encrypted data and encrypted key.

Step 6: Now tor URL can be generated which can be used for file (encrypted file) sharing on any system having tor environment.

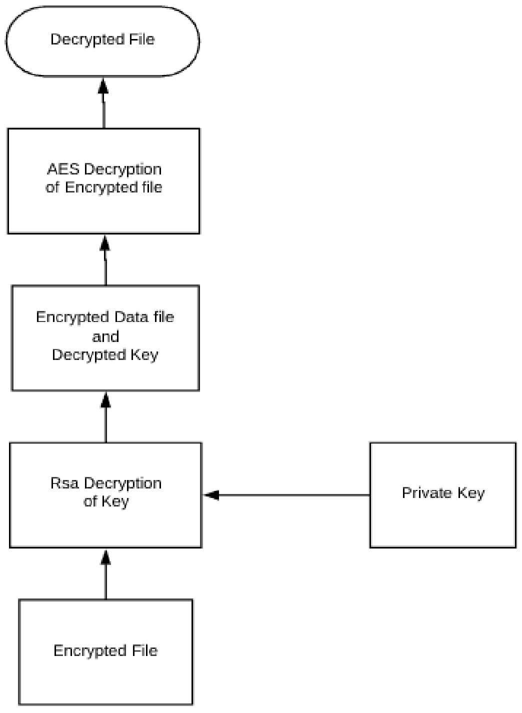
Step 7: To decrypt the file, receiver will use its private key which only he have, receiver can use onion link on tor browser and can download encrypted file. Now he can decrypt the file using its private key.

Step 8: Receiver have decrypted file without any security breach, while sharing the file between sender and receiver.

Encryption

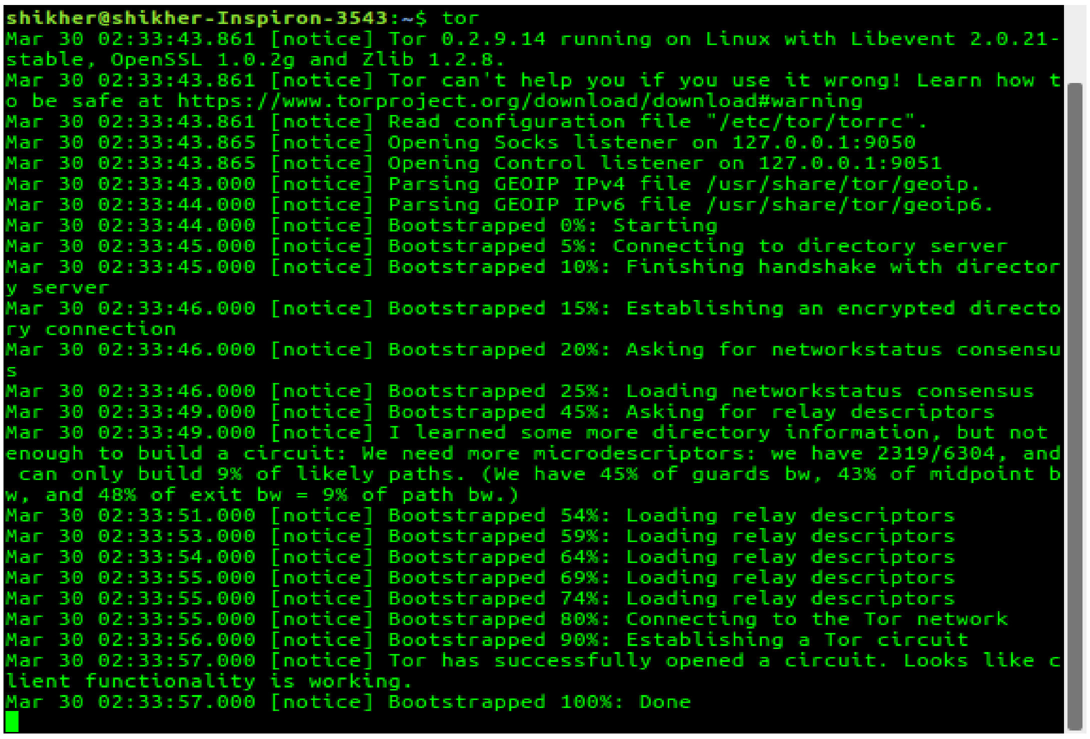


Decryption


Decrypted File
AES Decryption of Encrypted file
Encrypted Data file and Decrypted Key
Rsa Decryption of Key
Private Key
Encrypted File

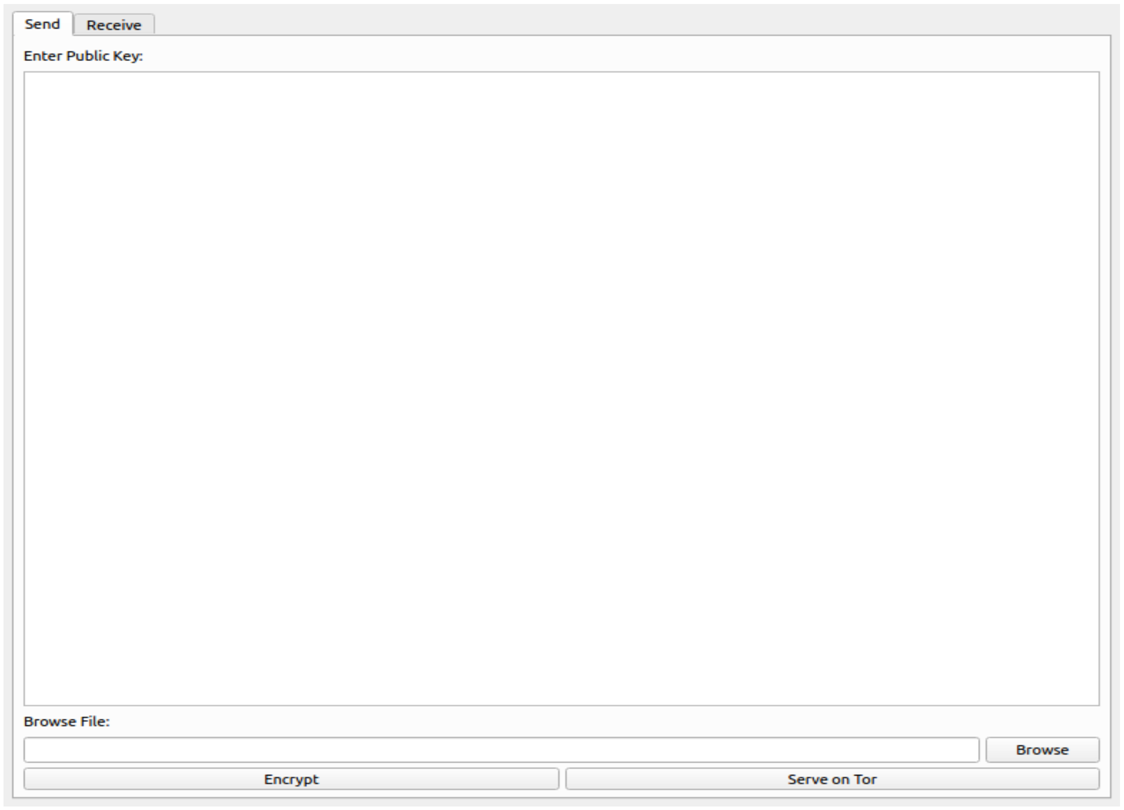
RESULTS AND OUTPUT

STARTING TOR SERVICE

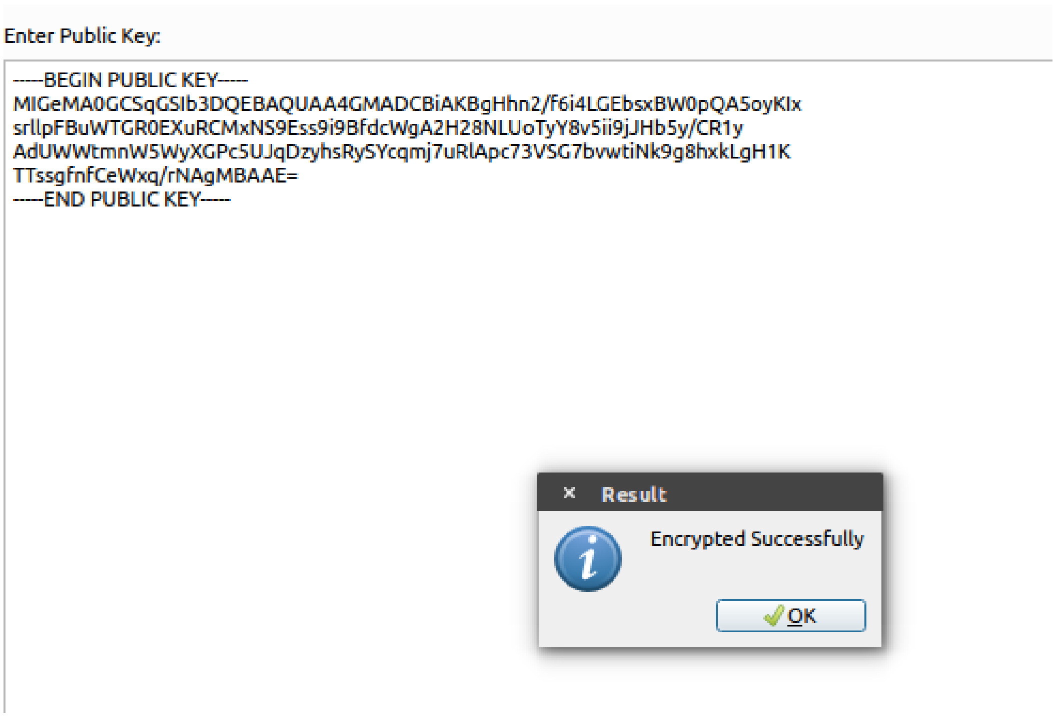


```
shikher@shikher-Inspiron-3543:~$ tor
Mar 30 02:33:43.861 [notice] Tor 0.2.9.14 running on Linux with Libevent 2.0.21-
stable, OpenSSL 1.0.2g and Zlib 1.2.8.
Mar 30 02:33:43.861 [notice] Tor can't help you if you use it wrong! Learn how t
o be safe at https://www.torproject.org/download/download#warning
Mar 30 02:33:43.861 [notice] Read configuration file "/etc/tor/torrc".
Mar 30 02:33:43.865 [notice] Opening Socks listener on 127.0.0.1:9050
Mar 30 02:33:43.865 [notice] Opening Control listener on 127.0.0.1:9051
Mar 30 02:33:43.000 [notice] Parsing GEOIP IPv4 file /usr/share/tor/geoip.
Mar 30 02:33:44.000 [notice] Parsing GEOIP IPv6 file /usr/share/tor/geoip6.
Mar 30 02:33:44.000 [notice] Bootstrapped 0%: Starting
Mar 30 02:33:45.000 [notice] Bootstrapped 5%: Connecting to directory server
Mar 30 02:33:45.000 [notice] Bootstrapped 10%: Finishing handshake with director
y server
Mar 30 02:33:46.000 [notice] Bootstrapped 15%: Establishing an encrypted directo
ry connection
Mar 30 02:33:46.000 [notice] Bootstrapped 20%: Asking for networkstatus consensu
s
Mar 30 02:33:46.000 [notice] Bootstrapped 25%: Loading networkstatus consensus
Mar 30 02:33:49.000 [notice] Bootstrapped 45%: Asking for relay descriptors
Mar 30 02:33:49.000 [notice] I learned some more directory information, but not
enough to build a circuit: We need more microdescriptors: we have 2319/6304, and
 can only build 9% of likely paths. (We have 45% of guards bw, 43% of midpoint b
w, and 48% of exit bw = 9% of path bw.)
Mar 30 02:33:51.000 [notice] Bootstrapped 54%: Loading relay descriptors
Mar 30 02:33:53.000 [notice] Bootstrapped 59%: Loading relay descriptors
Mar 30 02:33:54.000 [notice] Bootstrapped 64%: Loading relay descriptors
Mar 30 02:33:55.000 [notice] Bootstrapped 69%: Loading relay descriptors
Mar 30 02:33:55.000 [notice] Bootstrapped 74%: Loading relay descriptors
Mar 30 02:33:55.000 [notice] Bootstrapped 80%: Connecting to the Tor network
Mar 30 02:33:56.000 [notice] Bootstrapped 90%: Establishing a Tor circuit
Mar 30 02:33:57.000 [notice] Tor has successfully opened a circuit. Looks like c
lient functionality is working.
Mar 30 02:33:57.000 [notice] Bootstrapped 100%: Done
```

USER INTERFACE



ENCRYPT FILE



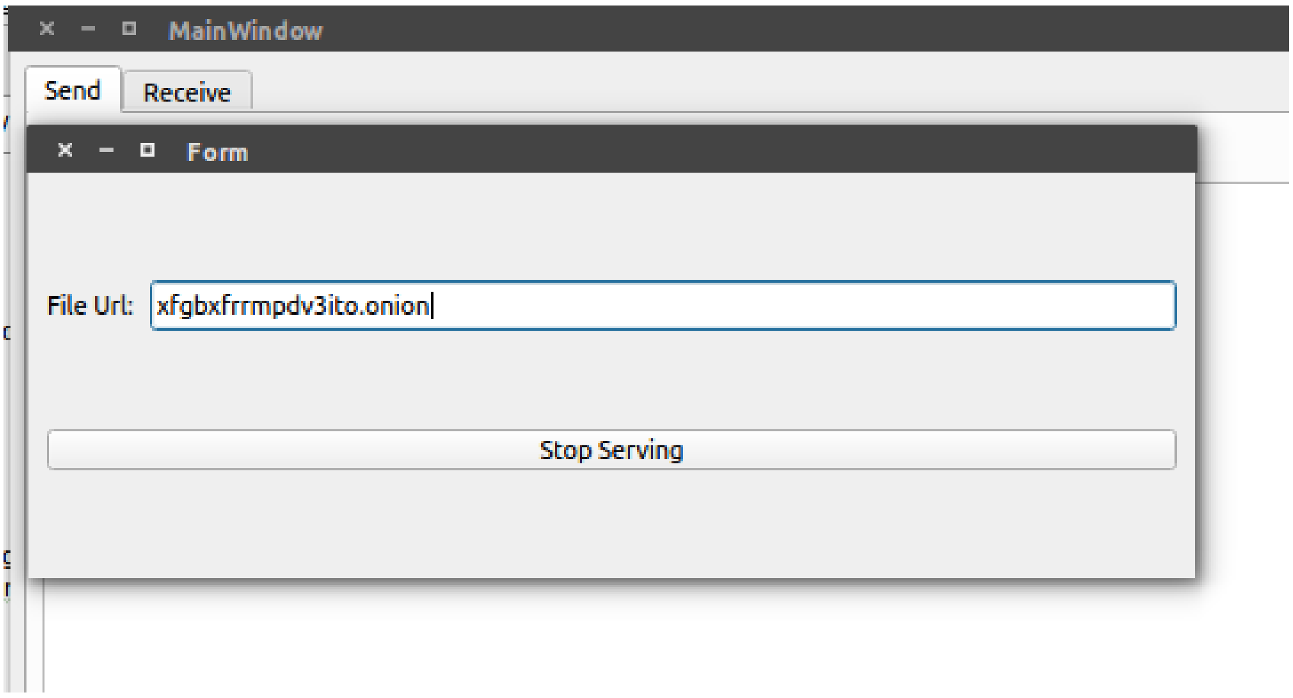
Enter Public Key:

```
-----BEGIN PUBLIC KEY-----
MIGeMA0GCSqGSIb3DQEBAQUAA4GMADCBiAKBgHhn2/f6i4LGEbsxBW0pQA5oyKIx
srllpFBuWTGR0EXuRCMxNS9Ess9i9BfdcWgA2H28NLUoTyY8v5ii9jJHb5y/CR1y
AdUWWtmnW5WyXGPc5UJqDzyhsRy5Ycqmj7uRlApc73VSG7bvwtiNk9g8hxkLgH1K
TTssgfnfCeWxq/rNAgMBAAE=
-----END PUBLIC KEY-----
```

Result

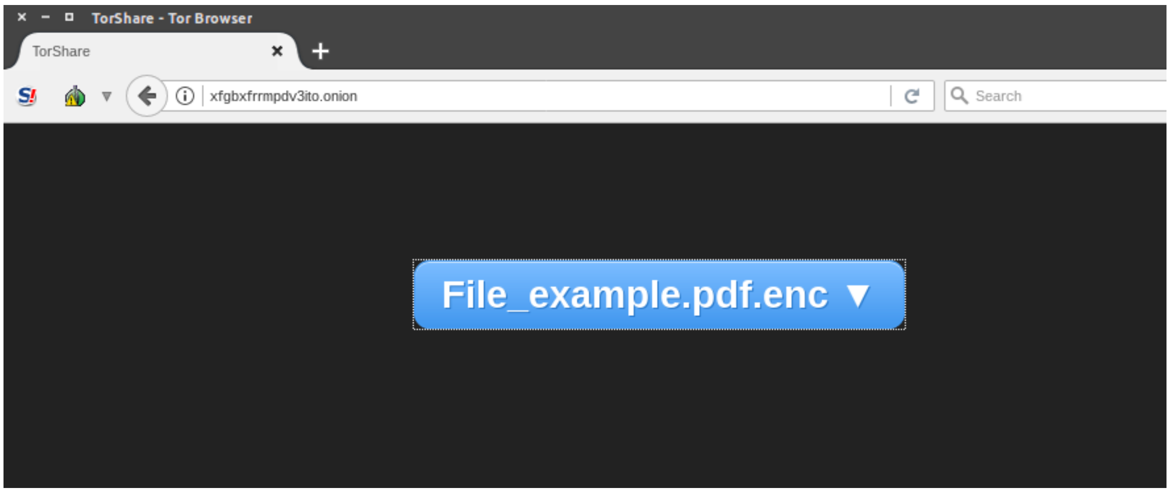
Encrypted Successfully

SERVE ON TOR

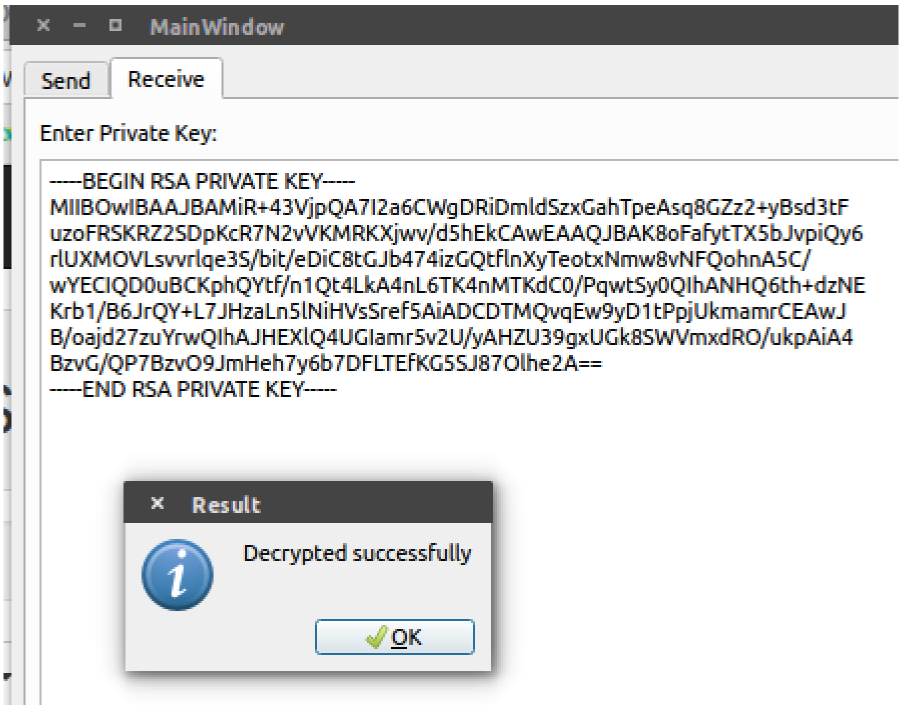


File Url: xfgbxfrrmpdv3ito.onion

DOWNLOAD FILE FROM TOR LINK



DECRYPT FILE



Enter Private Key:

```
-----BEGIN RSA PRIVATE KEY-----
MIIBOwIBAAJBAMiR+43VjpQA7I2a6CWgDRiDmldSzxGahTpeAsq8GZz2+yBsd3tF
uzoFRSKRZ2SDpKcR7N2vVKMRKXjwv/d5hEkCAwEAAQJBAK8oFafytTX5bJvpiQy6
rlUXMOVLsvvrlqe3S/bit/eDiC8tGJb474izGQtflnXyTeotxNmw8vNFQohnA5C/
wYEClQD0uBCKphQYtf/n1Qt4LkA4nL6TK4nMTKdC0/PqwtSy0QIhANHQ6th+dzNE
Krb1/B6JrQY+L7JHzaLn5lNiHVsSref5AiADCDTMQvqEw9yD1tPpjUkmamrCEAwJ
B/oajd27zuYrwQIhAJHEXlQ4UGIamr5v2U/yAHZU39gxUGk8SWVmxdRO/ukpAiA4
BzvG/QP7BzvO9JmHeh7y6b7DFLTEfKG5SJ87Olhe2A==
-----END RSA PRIVATE KEY-----
```

Decrypted successfully

FUTURE ENCHANCEMENTS

The future enhancement of the proposed Hybrid Encryption method involves securing online transaction in a wired environment, wireless environment and virtual network environment. The proposed hybrid encryption technique is also helpful in Mobile Banking and E- Banking. In virtual environment, data can be shared at two levels, such as Internet and Intranet. The proposed research work is used for sharing the secure data from one to others within the organization of the web server or virtual server.

In Local Area Network, the proposed hybrid encryption mechanism may be customized for transferring the sensitive data from work station to host based applications. In web-based applications, the proposed mechanism enables the transfer of sensitive data from user to user, from user to server and from server to server which are located outside of the organization. In a cloud environment, more number of people are accessing the web server locally or globally to share the sensitive data. The proposed hybrid encryption technique is very helpful to enhance the security for web-based transactions in future.

APPENDIX

CODE

ENCRYPT.PY

```
from Crypto.Cipher import AES, PKCS1_OAEP
from Crypto.PublicKey import RSA
from Crypto import Random


def aes_encrypt(data: bytes):
    random_gen = Random.new()
    key = random_gen.read(AES.block_size)
    iv = random_gen.read(AES.block_size)
    cipher = AES.new(key, AES.MODE_CFB, iv)
    msg = iv + cipher.encrypt(data)
    return key, msg


def rsa_encrypt(data: bytes, public_key: bytes):
    key = RSA.importKey(public_key)
    cipher = PKCS1_OAEP.new(key)
    ciphertext = cipher.encrypt(data)
    return ciphertext


def encrypt(public_key: bytes, file_path: str):
    try:
        with open(file_path, "rb") as f:
            data = f.read()
    except FileNotFoundError:
        return "File does not exist"

    key, enc_data = aes_encrypt(data)
    enc_key = rsa_encrypt(key, public_key)

    with open(file_path + ".enc", "wb") as f:
        f.write(enc_key)
        f.write(enc_data)

    return "Encrypted Successfully"
```

DECRYPT.PY

```python
import os

from Crypto.Cipher import AES, PKCS1_OAEP
from Crypto.PublicKey import RSA


def rsa_decrypt(data: bytes, private_key: RSA._RSAobj):
    cipher = PKCS1_OAEP.new(private_key)
    message = cipher.decrypt(data)
    return message


def aes_decrypt(key: bytes, data: bytes):
    iv = data[:AES.block_size]
    message = data[AES.block_size:]
    cipher = AES.new(key, AES.MODE_CFB, iv)
    return cipher.decrypt(message)


def decrypt(private_key: bytes, file_path: str):
    private_key = RSA.importKey(private_key)
    key_size = (private_key.size() + 1) // 8

    try:
        with open(file_path, "rb") as f:
            enc_key = f.read(key_size)
            enc_data = f.read()
    except FileNotFoundError:
        return "File does not exist"
    try:
        key = rsa_decrypt(enc_key, private_key)
    except ValueError:
        return "Could not decrypt the key"

    data = aes_decrypt(key, enc_data)

    with open(os.path.splitext(file_path)[0], "wb") as f:
        f.write(data)

    return "Decrypted successfully"
```

REFERENCES

- Tingyuan Nie, Chuanwang Song and Xulong Zhi, "Performance Evaluation of DES and Blowfish Algorithms", IEEE International Conference on Biomedical Engineering and Computer Science (ICBECS- 2010), pp. 1-4, 23-25 Apr 2010.

- Behrouz A Forouzan, “Data Communications and etworking”, cGraw-Hill, 4 th Edition.

- William Stallings, “Cryptography and etwork Security: Principles and Practice”, Pearson Education/Prentice Hall, 5 th Edition.

- E. Thambiraja, G. Ramesh and Dr. R. Umarani, "A Survey on Various Most Common Encryption Techniques", International Journal of Advanced Research in Computer Science and Software Engineering, Volume 2, Issue 7, pp. 226-233, July 2012.

- Diaa Salama Abdul. Elminaam, Hatem Mohamed Abdul Kader and Mohie Mohamed Hadhoud, "Performance Evaluation of Symmetric Encryption Algorithms", IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.12, pp. 280-286, December 2008.

- Uma Somani, Kanika Lakhani and Manish Mundra, “Implementing Digital Signatures with RSA Encryption Algorithm to Enhance the Data Security of Cloud in Cloud Computing”, 1st International Conference on Parallel, Distributed and Grid Computing (PDGC), pp. 211-216, 2010.

- Sriram Ramanujam and Marimuthu Karuppiah, "Designing an algorithm with high Avalanche Effect", IJCSNS International Journal of Computer Science and Network Security, VOL.11 No.1, pp. 106-111, January 2011.