

Ex9: Security

Lore

The Lizard Lords are displeased. The previously developed method of communication via TCP has been breached by humans and messages between Lizards have been intercepted. The Lizard Lords have decided they need a secure way to transmit messages between secret Lizard agents. This is your last chance to be spared the guillotine. You have been tasked with the development of a prototype for a secure communication channel that cannot be broken (not until quantum computers come along?) to demonstrate to the powers that be how humans may be stopped from making any use of intercepted messages.

Overview

The aim of this exercise is to familiarize you with asymmetrical encryption. In an asymmetric key encryption scheme, anyone can encrypt messages using the public key of the receiver, but only the receiver can decrypt because only they have access to the private key which is used to unroll the encryption. If the private key of some communication endpoint is obtained, any message pointed towards that endpoint can be decrypted if the public-private key algorithm is known.

Structure

To complete this exercise, you will be downloading the provided code skeletons and making use of the OpenSSL APIs to create public and private RSA keys in one process **A**. This process **A** shall communicate with a second process **B** via named pipes. Process **A** shall send its public key to process **B** using the pipe. Process **B** then encrypts a message taken via STDIN using this key and sends it back to the first process using another pipe. Then process **A** decrypts this message and displays it.

1. Use this website to generate RSA keys of size 2048 bits.
<https://8gwifi.org/RSAFunctionality?keysize=2048>
2. Save the public key generated into a text file “publicKey.txt” and private key in another text file “privateKey.txt”.
3. Take a screenshot of both the files containing the RSA keys side by side.
4. Create a named pipe “**pipeEx9**” using the following command.

```
$ mkfifo pipeEx9
```
5. In “**receiver.cpp**”, read the public and a private key from the text files and display the generated keys to the screen. Make use of the helper functions to read the keys. Then send its public key using the named pipe to the second program described in step 6.

- In “**sender.cpp**”, read the public key of the receiver using the named pipe and display it to the screen.
- sender.cpp** will take in a string message via STDIN and print it to the console after encryption. After the message is encrypted in sender.cpp, this message will be sent to receiver.cpp via the pipeEx9 created in step 4.
- receiver.cpp** will print the received encrypted message, decrypt it and print the decrypted message which should be the same as the initial message passed via STDIN to sender.cpp.
- Take a screenshot of the programs running side by side which should look like the picture below.

Three pre-implemented helper functions have been provided along with the templates. You can use these to assist in your coding or implement your own functions for reading and converting the keys.

- char* readKey(string fileName)** - reads the key from the text file and stores it in a char pointer
- RSA* convertPrivateKeyToRSA(FILE* fp)** - converts private key from opened file (FILE*) to RSA format
- RSA* convertPublicKeyToRSA(FILE* fp)** - converts public key from opened file (FILE*) to RSA format

You may need to read a char* as a FILE* to use the above functions, this can be done using POSIX string streams. https://www.gnu.org/software/libc/manual/html_node/String-Streams.html

When compiling with OpenSSL, include the necessary libraries as shown in the following command.

```
$ g++ source_code.cpp -lssl -lcrypto -o source_code
```

Submissions

You will submit the following at the end of this exercise on Canvas:

- C++ source file for receiver.cpp
- C++ source file for sender.cpp
- Makefile to compile the two programs
- Screenshot of public and private key text files described in the steps above.

```
(reptilian) 192.168.69.201
reptilian@localhost:~/ex9$ cat publicKey.txt
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAqN0qVzWH5EKKUt6Ylt1z
Naeo1EwNsnLqJ4fEBGP89sTVZEKqK2B024S0Iyo0k6xQ8reX3VclxVGERaiZPUuo
/GPUw7VwHGfCj0zlhGjsaDFNFDZDBzaDc5nbIkz0P6c6yIE/YZxVOXPqe2gMcFp
fLQbtKok4hh6/CkGwMzIAJjpmjptF107UxISkPttoa06WI/xYrfckH8PM3tBRJ9
mvXBF30vPe+gFwqGLfFWF3Vp+zYbKNS+Q5i0EPDn3XvmKilZ3j/T/DV4YAB38Eb
ls7h5WHT3F7zB230rBej836/v9IimWM3/RLZ96KKZcsAvGxkm3wBhN5CC7iSGNSy
GQIDAQAB
-----END PUBLIC KEY-----
reptilian@localhost:~/ex9$

(reptilian) 192.168.69.201
reptilian@localhost:~/ex9$ cat privateKey.txt
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAqN0qVzWH5EKKUt6Ylt1zNaeo1EwNsnLqJ4fEBGP89sTVZEKq
K2B024S0Iyo0k6xQ8reX3VclxVGERaiZPUuo/GPUw7VwHGfCj0zlhGjsaDFNFDZ
BzaDc5nbIkz0P6c6yIE/YZxVOXPqe2gMcFpflQbtKok4hh6/CkGwMzIAJjpmjpt
tF107UxISkPttoa06WI/xYrfckH8PM3tBRJ9mvXBF30vPe+gFwqGLfFWF3Vp+zY
bKNS+Q5i0EPDn3XvmKilZ3j/T/DV4YAB38EbLs7h5WHT3F7zB230rBej836/v9I
imWM3/RLZ96KKZcsAvGxkm3wBhN5CC7iSGNSyGQIDAQABAoIBAFMAKrlClaj+DKWR
P1JgVSG5YDXBkDX3X/rAM+wIja1pwI3Dl8LaZvYDe1uRgjHyY4UCdRk4o7ja4Wl
wnSrFmTz8pmj1BBPFu06CKU9CaenQlJrn0EiYUJV2K06XWJS04WUaeF7pLRRsp6
n3P1YnoSs2+Q6TqK1+CY0dxK2ILN+8YgoptqJPFnKcQfV292XxqN0UUVcUQLR4BEN
wBfSVWzEwKcNFq6U7SjohqhxRLIghIQPso8bil/r+h4v130uQxS778FZ6Ah2WJ
7LrBxqojt1l6zRs4I/UMzgz7JSKG/fzSIWfWm9gXcUz0L+LZ+F5e6BwQjQeLfcGq
Zpkm1DUCgYEA98HTFP4DzZbn4SzMijb1s4m2YJ20WqCpPq2n1Hn9GU2IjfnWwQn
n5CP1l/Qbi/AXpLHxHyWS5QWI7217gSq1aaq1Rjothc/kbPDMKHNFD0I4S/c7hS8
c4a1o0LEB0dRUr4znB9xWR9xYnsqTaD5jYmGgdZnxLeRi1w3RQARhMMcGyEARntL
vkv6QZSV3nVf7+/0tIilsr0Jnu3XkzvE7vWiic/d0gnvC1v5APx/CyIyHWiUw1a6
GZ0so7F8LYCg5xH0q8mSLbVKCIIMy2024bsE5Wexm60QTxYyp14KTW05oiI2uEzi
UEK7QHPIH0+mHEKVOqtmE13ZT4NIgQLBIZaez/MCgYEA1g0KmAmP5YxYEN6Jd2H+
Yza0dm1JLblyn2L0/56pUawP131Q/Eq3su0lQD09kwUEcXaCb7qn6w11I0nr3YKA
w3esTLGn9HI FhNVht3j93uP6WX9vWAC9NikoSxp6EaLlxQF/1GXDX0AIZj/+POj
x2RLIExxUWwxDNWuPcU0B3cGyANYChMjkaBAIL8qsLQAfUgYdLr+mvivYjUecCG
8Q2kuuqSgHBKzqIoES6RoG13E69G7J8XltqFMFX7AAFVtWfrmmg6WT0L9nZEnky2
wT9+1pQm+v1LEq6PTg7lyfz5euL9NLUjckNCFK/E7wILxIRjVIL6evezJGijUj07
/2/tQQKbgQDjTJYS7YsFGDYsF3aaGisd767iDCvoNaUmsCeU2TVoDlzUqCbG6qkz
wBPKAo7+osMQ5F24B50HTKMsPCABaDN95WpZWe1a/RaQ5/76w0FVlWnw7qHwsbdz
xMud0E56IDcWjh+ZxkJIOKkudLHG1D0t6oUxiX7Q/IxCzPea0AYZXA=
-----END RSA PRIVATE KEY-----
reptilian@localhost:~/ex9$
```

Figure 1: An example screenshot of the RSA key files

- Screenshot of the output from running the demo described in the steps above.

```

(reptilian) 192.168.69.201
reptilian@localhost:~/ex9$ ./receiver
Public key of receiver:
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAqN0qVzWH5EKKUt6Ylt1z
Naeo1EwNsnLqJ4fEBGP89sTVZEKqK2B024S0Iyo0k6xQ8reX3VclxVGERaiZPUuo
/GPUw7VwHGfCjOzLhgjsaDFNFDZDBzaDcD5nbIkz0P6c6yIE/YZxVOXPqe2gMcFp
fLQbtKoK4hh6/CKjGwMziAJjppjptF107UxISkPttoa06WI/xYrfckH8PM3tBRJ9
mvXBF30vWPe+gFwqGLfFWF3Vp+zYbKNS+Q5iOEPDn3XvmKiLZ3j/T/DV4YAB38Eb
ls7h5WHT3F7zB230rBej836/v9IimWM3/RLZ96KKZcsAvGxkm3wBhN5CC7iSGNSy
GQIDAQAB
-----END PUBLIC KEY-----

Private key of receiver:
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAqN0qVzWH5EKKUt6Ylt1zNaeo1EwNsnLqJ4fEBGP89sTVZEKq
K2B024S0Iyo0k6xQ8reX3VclxVGERaiZPUuo/GPUw7VwHGfCjOzLhgjsaDFNFDZ
BzaDcD5nbIkz0P6c6yIE/YZxVOXPqe2gMcFpfLQbtKoK4hh6/CKjGwMziAJjppj
ptF107UxISkPttoa06WI/xYrfckH8PM3tBRJ9mvXBF30vWPe+gFwqGLfFWF3Vp+z
YbKNS+Q5iOEPDn3XvmKiLZ3j/T/DV4YAB38Eb1s7h5WHT3F7zB230rBej836/v9I
imWM3/RLZ96KKZcsAvGxkm3wBhN5CC7iSGNSyGQIDAQABAoIBAFMAKrcLAj+DKWRk
P13gV5G5YDXBkD3X/rAm+wIja1pwI3Dl81aZvYDe1uRgjhXy4UCdRk4o7ja4Wl
wrSrfMt28pmj1BBPFu06CKKU9CaenQlJrN0EiYUJ2V2K06XJS04WUJaeF7pLRRsp6
n3P1YnoSs2+Q6TqK1+CY0dxK2ILn+8YqoptqJPFnkCqfV292XxqN0UVcUQLR4BEN
wBfSVmzEwKcNFq6U7SjohghRlIgiNHGQPso8biil/r+h4v130uQxS778F26Ah2WJ
7LrBxqojtil6zRs4I/UMzgz7J5Kg/fzSIWfwM9gXcUz0L+Lz+F5e6BwQj0eLFCqG
ZpkmiDUCgYEA98HTFP4DzZbn4SzMIjbs4m2YJ20WqCpPq2n1Hn9GU2IjfsnWWqN
n5CP1L/Qb1/AXplHxHyWSQWI7217gSqa1Rjtohc/kbPdmKHNFD0I4S/c7h58
c4a1o0LEB0dRur4rnB9xWR9xYnsqTaD5jYMggdZnxLeRi1w3RQARhMMcGYEARntL
vkv6Q2SV3nVf7+/0tIILsr0JnuJXkzvE7Wiiic/d0gnvC1v5APxC/yIyHWiuMLa6
GZSo5o7F8LYCg5xHQw8mSLbsVKCIMyz024bsE5Wexm60TxYyp14KTW05oI2uEzi
UEK7QhpIH0+mHEKv0qtmE13ZT4NIgQLBIzAeZ/MCgYEA1g0KmAmP5XyENI2d2H+
YzA0dmiJLbly2l0/56pUawP131Q/Eq3su0lqD09kwUEcXaCb7qn6wI1Ionr3YkA
w3esT1Gn9HI fHNVHnt3j93uP6W9vWAC9NikoSxp6EallxQf/1GXD0XAIzJ/+POj
x2RLU1ExxUWwxDNWuPcU0B3CgYANyChMjkaBAIL8qslQAfUgYdLR+mvIyUceCG
8Q2khuwqSgHbKzQIoES6RoG13E69G7J8XlTqfMFX7AAfVtWfmmg6WT0L9nEnky2
wT9+1pQm+v1LEq6PTrg7Lyfz5euL9MLUjckNCFK/E7wILxIRjVI16evezJGIluj07
/2/tQQKBgQDjTJYS7YsFGDYsf3aaG1sd7671DCvoNaUmsCeu2TVoDlZuqCbGxqWz
wBPKAo7+osMQSF24B50HTKMSPCABaDN95WpZwE1a/RaQs/76w0FVlWnw7qHwsbzd
xMud0E56IdcWjh+ZxkJI0KkudLHG1D0t6oUniX7Q/IxZcPea0AYZXA=
-----END RSA PRIVATE KEY-----

Encrypted Message:
C/.A.SS.Fp.i6i.8/B).C.C.%.%.%63NU.M."#MJD.
A.Ax./M.vq.ddFe.h.;Zr}U.
DA.w.w.
Decrypted message:
Hello world from Reptilian!
reptilian@localhost:~/ex9$

```

Figure 2: An example screenshot with output from running the solution.

NOTE 1: Use RSA_PKCS1_OAEP_PADDING for the padding mode for encryption and decryption (which takes 42 bytes).

NOTE 2: The maximum number of bytes you can encrypt for a 2048 bits modulus is 256 bytes – 42 bytes (for padding) = 214 bytes (so 213 total characters + 1 null terminator) when using RSA_PKCS1_OAEP_PADDING. Be mindful of null terminators.

Helpful Links

<https://www.ibm.com/docs/en/ztpf/2021?topic=concepts-public-key-cryptography>

<https://www.openssl.org/docs/man1.1.1/man1/rsa.html>

<https://medium.com/swlh/understanding-asymmetric-public-key-cryptography-24092bcd7741>

https://linux.die.net/man/3/rsa_size

https://linux.die.net/man/3/rsa_public_encrypt

https://linux.die.net/man/3/rsa_public_decrypt