

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 using one of the websites provided to generate public and private RSA keys. There will be two processes, the first 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 one of these websites to generate RSA keys of size 2048 bits.
<https://8gwifi.org/RSAFunctionality?keysize=2048> (or)
<https://travistidwell.com/jsencrypt/demo/>
2. Save the public key generated in 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.
6. In “**sender.cpp**”, read the public key of the receiver using the named pipe and display it.
7. **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.
8. **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.
9. 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.

1. **char* readKey(string fileName)** - reads the key from the text file and stores it in a char pointer
2. **RSA* convertPrivateKeyToRSA(FILE* fp, int length)** - converts private key in string format to RSA format
3. **RSA* convertPublicKeyToRSA(FILE* fp, int length)** - converts public key in string format to RSA format

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.

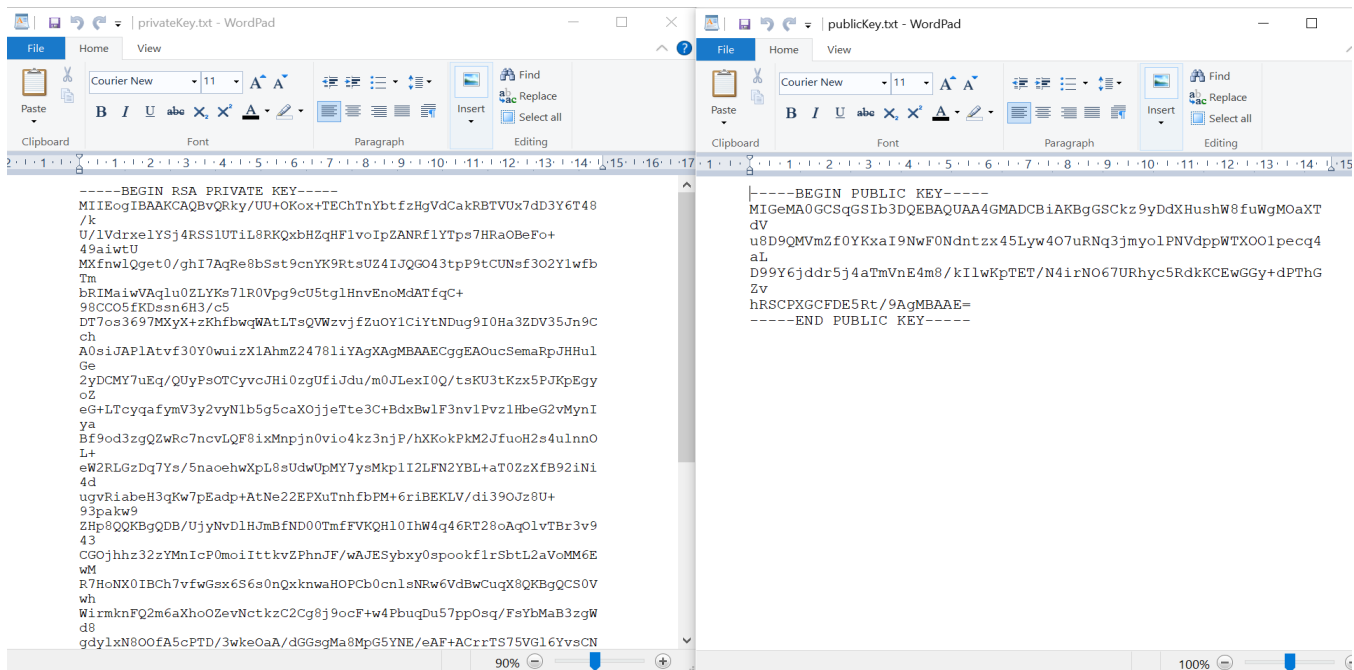


Figure 1: An example screenshot of the RSA key files

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

```
reptilian@localhost:~$ ./sender
Public key of receiver:
-----BEGIN RSA PUBLIC KEY-----
MIIBCAKCAQEAfF061D9XmGA0Hc05cwPtSnX212MkKwEvFZnXRwh+XiwXauuTCfrE
TTftnXVxpZ2kaTOX3MNC1t40FwsWnxG2pPp7i3eZs6+Hzrb08w69YI8U2Qgiof
sEdn4dyRt3u5ValYEt011vI6Y2GdAF6uHwhJsrPWR3WSCN4n516vePvjxsPATzR
0JpIRVH+eXAc7I1aSX5C5NoN8Lj0do1QfGQ0zk8wvK5aVP15S1nQaaGL+1/WqvS+A
AJARU0e6jc2bhlIR20TjZZ6cXYdC8iH0Eit9pkGfT6BU1/FHEA0TRbdQJ20HgZ1
LLFXMk13nSpVRnzjQCD0D8k/KDeA726hzwIBAw==
-----END RSA PUBLIC KEY-----

Enter the message:
Hello, OS students!
Encrypted Message:
???
'?'?'NNIS?e*?5??r1?~*???'g?<?d?Q??z??5???I?jE?q%?\|kMA
57j?e?
=?????S??{.7n=Nxf??D?8?''''??e?v)?#0/cM:??i??NNA
?<?%(?f?\?cVhm??y??JY?~x??' s(?f_??cP?h? eSk?E??LO?!?~*
reptilian@localhost:~$

reptilian@localhost:~$ ./receiver
Public key of receiver:
-----BEGIN RSA PUBLIC KEY-----
MIIBCAKCAQEAfF061D9XmGA0Hc05cwPtSnX212MkKwEvFZnXRwh+XiwXauuTCfrE
TTftnXVxpZ2kaTOX3MNC1t40FwsWnxG2pPp7i3eZs6+Hzrb08w69YI8U2Qgiof
sEdn4dyRt3u5ValYEt011vI6Y2GdAF6uHwhJsrPWR3WSCN4n516vePvjxsPATzR
0JpIRVH+eXAc7I1aSX5C5NoN8Lj0do1QfGQ0zk8wvK5aVP15S1nQaaGL+1/WqvS+A
AJARU0e6jc2bhlIR20TjZZ6cXYdC8iH0Eit9pkGfT6BU1/FHEA0TRbdQJ20HgZ1
LLFXMk13nSpVRnzjQCD0D8k/KDeA726hzwIBAw==
-----END RSA PUBLIC KEY-----

Private key of receiver:
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAfF061D9XmGA0Hc05cwPtSnX212MkKwEvFZnXRwh+XiwXauuT
CfrETTftnXVxpZ2kaTOX3MNC1t40FwsWnxG2pPp7i3eZs6+Hzrb08w69YI8U2Q
giofEEdn4dyRt3u5ValYEt011vI6Y2GdAF6uHwhJsrPWR3WSCN4n516vePvjxsP
AtzR0JpIRVH+eXAc7I1aSX5C5NoN8Lj0do1QfGQ0zk8wvK5aVP15S1nQaaGL+1/Wq
vS+AAJARU0e6jc2bhlIR20TjZZ6cXYdC8iH0Eit9pkGfT6BU1/FHEA0TRbdQJ20
HgZ1LLFXMk13nSpVRnzjQCD0D8k/KDeA726hzwIBAwKCAQ09k3yYf4+66s1pM3ui
Ap4xo/m6Qh8K3S5ETovWV7pcrPHR7db/ILEJUKTo4/Dvm2bdq7L0iPPS1ksqBU
tnnDUaeyT7vNH6+Zz03CiCd0VtLHdWshBUgLSqWkwPp9DjxjQ30Fi059tGWYRN
V1HQmtvMd+Qvo7aw1BqZ1HT7CQ7c0gstZMHqPAMqJ5AJmtuKfYSh2bw8XiPowko
/j3UGWT+XKMhQ8xuk60u2Fu88KUtRKdQEGkMk76kXaja8PYGEAJd3sWHD7+4QwC
GA7U071M04JngM+S+RpZ/YZruJC77KfaJZZ77vpQTBs7bn77+xoZBdi+b1m8bYM
KzbAa0BAP08XB/rTSQ120D1+NiDw+10S0DuSg08w+1W8Ejc10575UfpZZYXsj
V4dXWUShHNS8dyo+K1E77yYf7xgr61E068YwsEC0Pc1Lc2AP8s/QMJC846Km15
Smt15Ymm3DZ11fYEN5Fw/h5dJh/xandGfCH11U12hVKkg+G7iHdJA0GBAmeBKfi+
faB3F3+RrGLYKID8021rosRsTpyDhSPUW9d3BwpvG2d0kUe409Jr3ShwB4wG
6+PzZMbEO4DktIE2zNVcUxSWi9oCoPm5A0MnMn6q4D0XALsCMTQVj1UEpc25JMJS
1nIBzMFcsHBH4IDEeN6t9XuETVDrSQ4hpVhXA0GBAKE1+PwqdI3hgF54CZUJOTKp
40MPXBMV4BLUFkoDCTOCmn7i/xSQ66dtK1o604mWEZ7twhFUGwmn8H1qn2VynDY
jR91LRmtWu09sHokAKMcqkysBR062BQ3Ee71vEks7uY/IYJQugqMmTbr/21vou
/WvSDj05rjchAp29BtAc0BAtUxaq+/-mr3PvXUlyEHLc0FpSkZhwDtf70hxr
h1s+T6BL0Tz0+JcAU14cvH6Mng00EnUKY08tF02YeFk1149j1hKXtWbMKZ7
V41a1JR6tNkqydwY3g0XUNYb0kmGIMZEwBMyyUITFSV70CC+z8e0et10Cc217B
GOWPA0GBAMQeFUXpIDMU11aZs5XuwU1HNEHh8sSD83XB85jbPTXD4z9Hnqm1uR
BidPMp90sjFLu5Zj04u5adh5m7a1goukzuh/OKp35fAtaMufvxdV8jW7mwOKp1
IeJJ615MnSI0Knt1FNCqpsYX0iON08cPdfS7j5Ua10gF1/I03DEb
-----END RSA PRIVATE KEY-----

Encrypted Message:
???
'?'?'NNIS?e*?5??r1?~*???'g?<?d?Q??z??5???I?jE?q%?\|kMA
57j?e?
=?????S??{.7n=Nxf??D?8?''''??e?v)?#0/cM:??i??NNA
?<?%(?f?\?cVhm??y??JY?~x??' s(?f_??cP?h? eSk?E??LO?!?~*
Decrypted message:
Hello, OS students!
reptilian@localhost:~$
```

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/support/knowledgecenter/en/SSB23S_1.1.0.13/gtps7/s7pkey.html

<https://www.openssl.org/docs/man1.0.2/man3/rsa.html>

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