Project 1 (You may work on it individually or in a team of TWO members)
Submission: submit the three programs to Blackboard and upload them on the Virtual Machine under your home directory.

Evaluation: your work is evaluated according to (1) the quality of your accomplishment presented and demonstrated in class, (2) the quality of your presentation and demonstration, and (3) the quality of your answers to the questions from the instructor and students during the presentation. The programs that cannot be compiled or executed may receive up to 5% of the total points.

Deadlines: Due to the nature of the project, NO submission is accepted after the deadline.

- Submit your **programs** and **PPT slides** on Blackboard by **11:59pm Oct 2**. Every team member needs to submit a copy of all the programs and PPT Slides.
- Present and demonstrate your work on **Oct 3 (Tuesday)** in class.

Your choice of option in Task I: Please POST your choice of option that you are going to work on for Project 1 on "Discussion Board" on Blackboard. Each team only needs to make one post. In your post, please include (1) your name and your collaborator's name if any, and (2) your choice of option including the option description. Each option is allowed to be chosen by **up to 5 individuals or teams**. **First come, first serve.**

Task I: Write THREE independent programs in three SEPARATE directories: a ***key generation*** program, a ***sender's*** program, and a ***receiver's*** program, to implement ONE of the following options of simplified cryptographic applications.

- In the ***key generation*** program (it is required for EACH of the following options) in the directory "**KeyGen**",
  1. Create a pair of RSA public and private keys for **X**, $\boldsymbol{Kx}^+$ and $\boldsymbol{Kx}^-$;
  2. Create a pair of RSA public and private keys for **Y**, $\boldsymbol{Ky}^+$ and $\boldsymbol{Ky}^-$;
  3. Get the modulus and exponent of each RSA public or private key and save them into files named "*XPublic.key*", "*XPrivate.key*", "*YPublic.key*", and "*YPrivate.key*", respectively;
  4. Take a 16-character user input from the keyboard and save this 16-character string to a file named "*symmetric.key*". This string's 128-bit UTF-8 encoding will be used as the 128-bit AES symmetric key, ***Kxy***, in your application.

Option 1: Public-key encrypted message and its authentic digital digest

- In this option, **X** is the **sender** and **Y** is the **receiver**.
- In the ***sender's*** program in the directory "**Sender**", calculate **RSA-En** $_{Ky+}$ (**AES-En** $_{Kxy}$ (**SHA256** (M)) || M)
  1. To test this program, the corresponding key files need to be copied here from the directory "KeyGen"
  2. Read the information on the keys to be used in this program from the key files and generate $\boldsymbol{Ky}^+$ and ***Kxy***.
  3. Display a prompt "`Input the name of the message file:`" and take a user input from the keyboard. This user input provides the name of the file containing the message **M**. **M** can NOT be assumed to be a text message. The size of the message M could be much larger than 32KB.
  4. Read the message, *M*, from the file specified in Step 3 piece by piece, where each piece is recommended to be a small multiple of 1024 bytes, calculate the SHA256 hash value (digital digest) of the entire message M, i.e., **SHA256**(M), SAVE it into a file named "*message.dd*", and DISPLAY **SHA256**(M) in Hexadecimal bytes.
  5. Calculate the **AES Encryption** of **SHA256**(M) using ***Kxy*** (NO padding is allowed or needed here. Question: how many bytes are there in total?), SAVE this AES cyphertext into a file named "*message.add-msg*", and DISPLAY it in Hexadecimal bytes. APPEND the message M read from the file specified in Step 3 to the file "*message.add-msg*" piece by piece.
  6. Calculate the **RSA Encryption** of (**AES-En** $_{Kxy}$ (**SHA256** (M)) || M) using $\boldsymbol{Ky}^+$ by reading the file "*message.add-msg*" piece by piece, where each piece is recommended to be 117 bytes if "RSA/ECB/PKCS1Padding" is used. (Hint: if the length of the last piece is less than 117 bytes, it needs to be placed in a byte array whose array size is the length of the last piece before being encrypted.) SAVE the resulting blocks of RSA ciphertext into a file named "*message.rsacipher*".
- In the ***receiver's*** program in the directory "**Receiver**", using **RSA** and **AES Decryptions** to get S**HA256** (M) and M, compare **SHA256**(M) with the locally calculated SHA256 hash of M, report hashing error if any, and then save M to a file.
  1. To test this program, the corresponding key files need to be copied here from the directory "KeyGen", and the file "*message.rsacipher*" needs to be copied here from the directory "Sender".
  2. Read the information on the keys to be used in this program from the key files and generate $\boldsymbol{Ky}^-$ and ***Kxy***.
  3. Display a prompt "`Input the name of the message file:`" and take a user input from the keyboard. The resulting message M will be saved to this file at the end of this program.

4 Read the ciphertext, C, from the file "*message.rsacipher*" block by block, where each block is recommended to be 128 byte long if "RSA/ECB/PKCS1Padding" is used. Calculate the **RSA Decryption** of **C** using $\boldsymbol{Ky}^-$ block by block to get **AES-En** $_{Kxy}$ (**SHA256** (M)) || M, and save the resulting pieces into a file named "*message.add-msg*".

5 Read the first 32 bytes from the file "*message.add-msg*" to get the ***authentic digital digest*** **AES-En** $_{Kxy}$ (**SHA256** (M)), and copy the message **M**, i.e., the leftover bytes in the file "*message.add-msg*", to a file whose name is specified in Step 3. (Why 32 bytes? Why is the leftover M?) Calculate the **AES Decryption** of this authentic digital digest using ***Kxy*** to get the ***digital digest*** **SHA256**(M), SAVE this digital digest into a file named "*message.dd*", and DISPLAY it in Hexadecimal bytes.

6 Read the message ***M*** from the file whose name is specified in Step 3 piece by piece, where each piece is recommended to be a small multiple of 1024 bytes, calculate the SHA256 hash value (digital digest) of the entire message M, compare it with the digital digest obtained in Step 5, and display whether the digital digest passes the authentication check.

Option 2: symmetric-key encrypted message and its digital signature

- In this option, **X** is the **sender** and **Y** is the **receiver**.
- In the ***sender's*** program in the directory "**Sender**", calculate **AES-En** $_{Kxy}$ (**RSA-En** $_{Kx-}$ (**SHA256** (M)) || M)
  1 To test this program, the corresponding key files need to be copied here from the directory "KeyGen"
  2 Read the information on the keys to be used in this program from the key files and generate $\boldsymbol{Kx}^-$ and ***Kxy***.
  3 Display a prompt "`Input the name of the message file:`" and take a user input from the keyboard. This user input provides the name of the file containing the message **M**. **M** can NOT be assumed to be a text message. The size of the message M could be much larger than 32KB.
  4 Read the message, *M*, from the file specified in Step 3 piece by piece, where each piece is recommended to be a small multiple of 1024 bytes, calculate the SHA256 hash value (digital digest) of the entire message M, i.e., **SHA256**(M), SAVE it into a file named "*message.dd*", and DISPLAY **SHA256**(M) in Hexadecimal bytes.
  5 Calculate the **RSA Encryption** of **SHA256**(M) using $\boldsymbol{Kx}^-$ (Question: how many bytes is the cyphertext?), SAVE this RSA cyphertext (the digital signature of **M**), into a file named "*message.ds-msg*", and DISPLAY it in Hexadecimal bytes. APPEND the message M read from the file specified in Step 3 to the file "*message.ds-msg*" piece by piece.
  6 Calculate the **AES Encryption** of (**RSA-En** $_{Kx-}$ (**SHA256** (M)) || M) using ***Kxy*** by reading the file "*message.ds-msg*" piece by piece, where each piece is recommended to be a multiple of 16 bytes long. (Hint: if the length of the last piece is less than that multiple of 16 bytes, it needs to be placed in a byte array whose array size is the length of the last piece before being encrypted.) SAVE the resulting blocks of AES ciphertext into a file named "*message.aescipher*".
- In the ***receiver's*** program in the directory "**Receiver**", using **AES** and **RSA Decryptions** to get S**HA256** (M) and M, compare **SHA256**(M) with the locally calculated SHA256 hash of M, report hashing error if any, and then save M to a file.
  1 To test this program, the corresponding key files need to be copied here from the directory "KeyGen", and the file "*message.aescipher*" needs to be copied here from the directory "Sender".
  2 Read the information on the keys to be used in this program from the key files and generate $\boldsymbol{K_x}^+$ and ***Kxy***.
  3 Display a prompt "`Input the name of the message file:`" and take a user input from the keyboard. The resulting message M will be saved to this file at the end of this program.
  4 Read the ciphertext, **C**, from the file "*message.aescipher*" block by block, where each block needs to be a multiple of 16 bytes long. (Hint: if the length of the last block is less than that multiple of 16 bytes, it needs to be placed in a byte array whose array size is the length of the last piece before being decrypted.) Calculate the **AES Decryption** of **C** using ***Kxy*** block by block to get **RSA-En** $_{Kx-}$ (**SHA256** (M)) || M, and save the resulting pieces into a file named "*message.ds-msg*".
  5 If using "RSA/ECB/PKCS1Padding", read the first 128 bytes from the file "*message.ds-msg*" to get the ***digital signature*** **RSA-En** $_{Kx-}$ (**SHA256** (M)), and copy the message **M**, i.e., the leftover bytes in the file "*message.ds-msg*", to a file whose name is specified in Step 3. (Why 128 bytes? Why is the leftover M?) Calculate the **RSA Decryption** of this digital signature using $\boldsymbol{K_x}^+$ to get the ***digital digest*** **SHA256**(M), SAVE this digital digest into a file named "*message.dd*", and DISPLAY it in Hexadecimal bytes.
  6 Read the message ***M*** from the file whose name is specified in Step 3 piece by piece, where each piece is recommended to be a small multiple of 1024 bytes, calculate and display the SHA256 hash value (digital digest) of the entire message M, compare it with the digital digest obtained in Step 5, display whether the digital digest passes the authentication check.

Option 3: Digital envelope including keyed hash MAC

- In this option, **X** is the **sender**, **Y** is the **receiver**, ***Kxy*** is the random symmetric key generated by X.
- In the ***sender's*** program in the directory "**Sender**", calculate S**HA256** (***Kxy*** || M || ***Kxy***), **AES-En** $_{Kxy}$ (M), and **RSA-En** $_{Ky+}$ (***Kxy***)

1 To test this program, the corresponding key files need to be copied here from the directory "KeyGen"
2 Read the information on the keys to be used in this program from the key files and generate $\boldsymbol{Ky}^+$ and ***Kxy***.
3 Display a prompt "`Input the name of the message file:`" and take a user input from the keyboard. This user input provides the name of the file containing the message **M**. **M** can NOT be assumed to be a text message. The size of the message M could be much larger than 32KB.
4 WRITE ***Kxy*** to a file named "*message.kmk*", read the message **M** from the file specified in Step 3 piece by piece, APPEND M to the file "*message.kmk*", and finally APPEND ***Kxy*** to the file "*message.kmk*".
5 Read ***Kxy*** || M || ***Kxy*** piece by piece from the file "*message.kmk*", where each piece is recommended to be a small multiple of 1024 bytes, calculate the ***keyed hash MAC***, i.e., the **SHA256** hash value of (***Kxy*** || M || ***Kxy***), SAVE this keyed hash MAC into a file named "*message.khmac*", and DISPLAY it in Hexadecimal bytes.
6 Calculate the **AES Encryption** of **M** using ***Kxy*** by reading the file specified in Step 3 piece by piece, where each piece is recommended to be a multiple of 16 bytes long. (Hint: if the length of the last piece is less than that multiple of 16 bytes, it needs to be placed in a byte array whose array size is the length of the last piece before being encrypted.) SAVE the resulting blocks of AES ciphertext into a file named "*message.aescipher*".
7 Calculate the **RSA Encryption** of ***Kxy*** using $\boldsymbol{Ky}^+$. (Hint: if "RSA/ECB/PKCS1Padding" is used, what is the length of the resulting ciphertext?) SAVE the resulting RSA ciphertext into a file named "*kxy.rsacipher*".

- In the ***receiver's*** program in the directory "**Receiver**", using **RSA Decryption** to get ***Kxy***, **AES Decryptions** to get **M**, compare **SHA256**(***Kxy*** || **M** || ***Kxy***) with the locally calculated SHA256 hash of (***Kxy*** || **M** || ***Kxy***), and report hashing error if any.
  1 To test this program, the corresponding key files need to be copied here from the directory "KeyGen", and the files "*message.khmac*", "*message.aescipher*", and "*kxy.rsacipher*" need to be copied here from the directory "Sender". (Hint: the file "*symmetric.key*" should NOT be copied here from the directory "KeyGen".)
  2 Read the information on the keys to be used in this program from the key file and generate $\boldsymbol{Ky}^-$.
  3 Display a prompt "`Input the name of the message file:`" and take a user input from the keyboard. The resulting message M will be saved to this file at the end of this program.
  4 Read the ciphertext, **C1**, from the file "*kxy.rsacipher*". Calculate the **RSA Decryption** of **C1** using $\boldsymbol{Ky}^-$ to get the random symmetric key ***Kxy***, SAVE ***Kxy*** to a file named "*message.kmk*", and DISPLAY ***Kxy*** in Hexadecimal bytes.
  5 Read the ciphertext, **C2**, from the file "*message.aescipher*" block by block, where each block needs to be a multiple of 16 bytes long. (Hint: if the length of the last block is less than that multiple of 16 bytes, it needs to be placed in a byte array whose array size is the length of the last piece before being decrypted.) Calculate the **AES Decryption** of **C2** block by block using the ***Kxy*** obtained in Step 4 to get **M**, WRITE the resulting pieces of **M** into the file specified in Step 3, and also APPEND those resulting pieces of M to the file "*message.kmk*" created in Step 4. Finally APPEND ***Kxy*** after **M** to the file "*message.kmk*". (Hint: at the end of this Step, ***Kxy*** || **M** || ***Kxy*** is written to the file "*message.kmk*", and **M** is written to the file specified in Step 3.)
  6 Read ***Kxy*** || M || ***Kxy*** piece by piece from the file "*message.kmk*", where each piece is recommended to be a small multiple of 1024 bytes, calculate the ***keyed hash MAC***, i.e., the **SHA256** hash value of (***Kxy*** || M || ***Kxy***), COMPARE this keyed hash MAC with the keyed hash MAC read from the file "*message.khmac*", DISPLAY whether it passes the message authentication checking, and DISPLAY both keyed hash MACs in Hexadecimal bytes.

Task II: Test your programs on the Virtual Servers in the cloud.

1. MAKE a directory "**Project1**" under your home directory on cs3750a.msdenver.edu and cs3750b.msudenver.edu, and three subdirectories "**KeyGen**", "**Sender**", and "**Receiver**" under "**Project1**".
2. UPLOAD and COMPILE the ***key generation*** program under "Project1/KeyGen" on **cs3750a**.msudenver.edu, the ***sender's*** program under "Project1/Sender" **cs3750a**.msudenver.edu, and the ***receiver's*** program under "Project1/Receiver" on **cs3750b**.msudenver.edu.
3. You may the small text file CS3700.htm, the medium-sized non-text file HW1_CS3750, and the large file 01_Intro.pdf for testing. TEST your programs for all the possible cases including the cases where there isn't a message authentication error and the cases where there is a message authentication error.

Task III: Present your work and demo your programs on the Virtual Servers in class.

1. Using the message file provided by the instructor on the day of presentation to demo all of your three programs step by step. Show the outputs and the files generated by your programs as your programs run.
2. If you are working in a team of two, the ***sender's*** program and the ***receiver's*** program must be demoed by two students in their own home directories, respectively. The instructor will help copy the files in between.