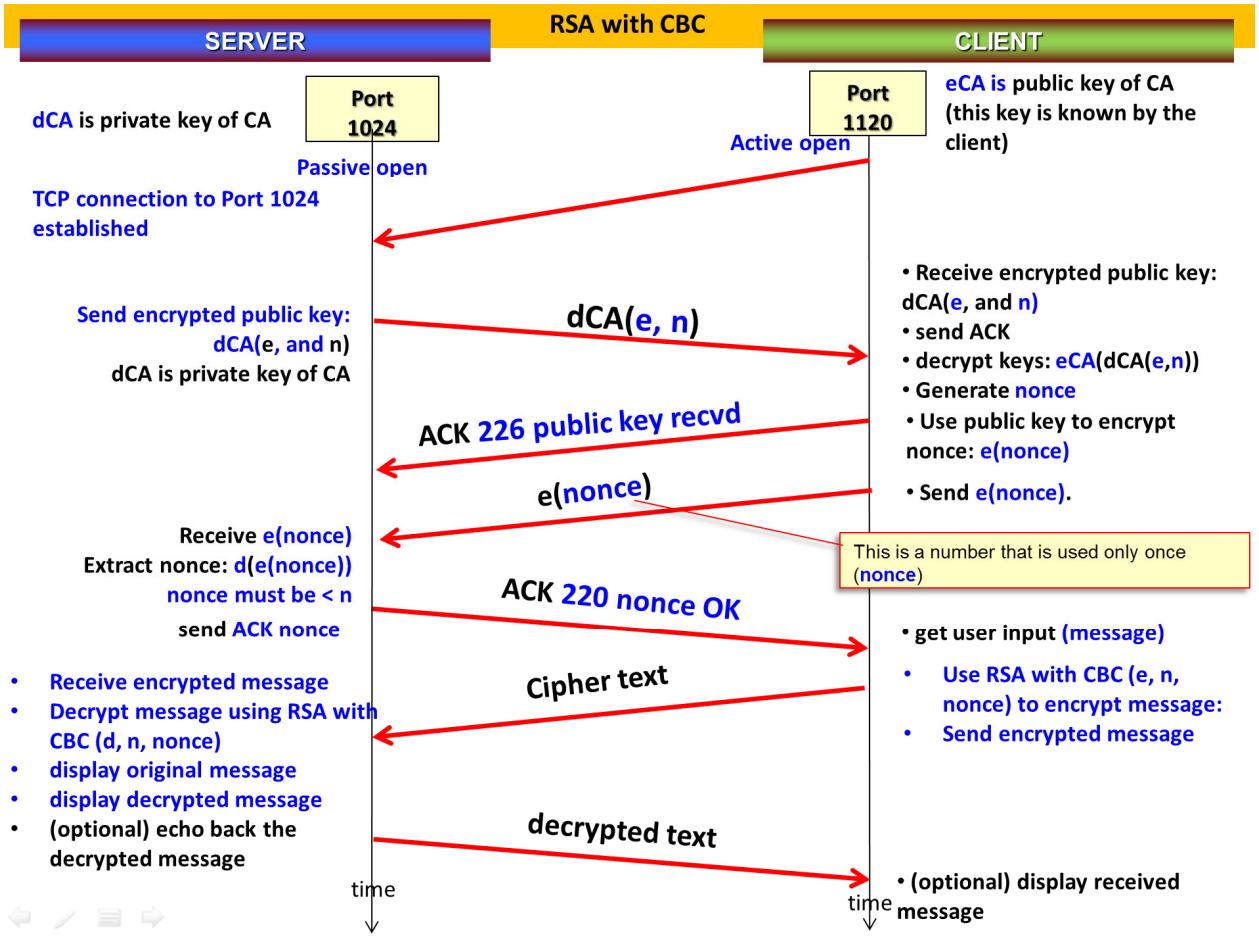
Operating Systems and Networks 159.342 Assignment 2 due 17 May

Cryptography: RSA algorithm with Cipher Block Chaining

Secure Communication Protocol

In this assignment your task is to implement a hybrid encryption algorithm called **RSA with Cipher Block Chaining** for communicating TCP client-server applications. In the TCP client-server examples, as you might remember, a client types a message, and then sends it to a server. In turn, the server echoes back the message. In this assignment, you have to modify the client so that it ***encrypts*** all messages sent to the server. In addition, you also have to modify the server, so that it ***decrypts*** *the encrypted* message locally.

Below are the basic steps describing the secure communication protocol.



Create a fixed RSA private key (**dCA**) and public key (**eCA**) for a **CA (certification authority)**.
Assume that the CA issued a certificate for a server, containing its public key. That means the server has **dCA(e,n)**. This is an encrypted public key of the server. Assume further that the client knows the public key of the certification authority (**eCA**), who issued the certificate.

As always, the server must be up and running prior to a client connecting to it.
After the server accepts a client connection request, establishing a TCP connection, the server must send the client its **encrypted *public key* dCA(e,n)**. In turn, the client extracts the public key using its copy of the certification authority's public key: **eCA(dCA(e,n)),** then ACKs the receipt of the key**.** Next**,** the client informs the server of the random number (**nonce**) that will be used for the initial encryption and decryption operations. This nonce is sent encrypted using **e(nonce)**. Successive messages coming from the client will then be encrypted accordingly using RSA-CBC, and the server will decrypt the received messages using RSA-CBC. The server may also echo back to the client the decrypted messages (this is optional - not required).

The client/server must be able to show the original, encrypted and decrypted messages, as seen in the example below:

- The client types: hello

- The server prints locally: The received encrypted message was: 10898 15630 8308 321 13772 22674 22040

After decryption, the message found is: hello

Design issues

There are many details in the implementation of a real RSA with Cipher Block Chaining that might have to be left out in order to keep it under the scope of an assignment. You have to ask the following questions regarding some decisions about the implementation:

a) What are the sizes of the keys? (aim for a big number(in the thousands) for the variable **n** in **RSA**)
b) Should the encryption be done character by character, or by block of characters?
c) Should padding be used?
d) Do you need an arbitrary precision library, or the keys are small enough to use a simple exponential code?
e) How are the keys sent to the client (your protocol design should specify: format, message order, etc).

The answers to these questions will establish how your own "encryption protocol" works.

For more details, check out the following documents:
1) The details of the core algorithm that must be implemented can be found in "**Lecture_2022_07_Network_Security_Implementing RSA_CBC.pptx**" You are allowed to use relatively small prime number pairs, but do not use pairs that will generate the same original message after encryption.
2) Use the **TCP client/server (IPV4, 1PV6), windows codes** as start-up codes (found in our 159342 Stream website). Modify these source codes for your implementation of the secure communication protocol.

Testing the assignment:

For quickly testing the assignment, use the batch file, run.bat provided.

The assignment is going to be marked based on *functionality and design*.

The marks are distributed as follows:

- **2** marks: for the successful sending of the encrypted RSA public key to the client, and decryption of the public key by the client.
- **2** marks: for correctly sending the encrypted random number (**encrypted**(**nonce**)) to the server. That also includes the server decrypting and extracting the **nonce** correctly.
- **8** marks: for the correct implementation of RSA with Cipher Block Chaining. The encryption/decryption results should always be correct for any ASCII characters. Make sure that the combination of the **selected RSA keys** and **nonce** will not have any problem encrypting and decrypting messages.
- **3** marks: **Documentation** (MS Word). Submit an accomplished **Checklist** (fill-up the given check list form that comes with this document).
  Write a brief description of your system (i.e. specify the message format, encoding, decoding scheme, how keys are sent, generation of keys, etc.).
  Provide snapshots of three complete client/server interactions, using the following test input strings:

  For testing, use the following test input strings:
  1. "**the quick brown fox jumps over the lazy dog**."
  2. "**AAA**"
  3. "**555**"

Also, indicate if you have implemented the Euclidean Algorithm and the Extended Euclidean Algorithm in combination with a big number library, to get some bonus marks.

***Bonus marks (max. of 2 marks)** will be given to those who can implement and utilise correctly the boost

library's big number support, allow for very large public and private keys (make a note in your assignment submission if you did this. **Boost**: https://www.boost.org/

Also include instructions on how to install the big number library you have utilised.

Notes:

1 - Submit your files electronically via Stream, in zip format. Submit everything, including the folders, make files, batch file, project support files, check list, snap shots and brief documentation of your system. If you are using a special big number library, you should submit that, too. Everything should compile in a Windows platform.
2 - This assignment is worth **15 marks (**+ bonus, if your submission qualifies**)**.
3 - Marks will be subtracted for obvious copying and/or for delays without justification.

Checklist (Include this in your submission)
Please accomplish the following check list in order to allow for accurate marking of your assignment.

	Item	your assignment details		Comments
1	Names and ID numbers of Group Members			(Maximum of 3 members in a group)
2	Operating System(s) used for testing your client and server codes	Windows 10		
3	Compiler used	g++ 11.2.0		g++ 11.2.0 is required
4	IDE used			
5	Required Functionalities	Sending encrypted RSA public key to the client, and decryption of the public key by the client	full/partial/none	Indicate '**full**', if you have completed the implementation of the required command, '**partial'**, if you are only submitting a partial implementation, or '**none**', if not accomplished.
		Sending encrypted nonce to the server, and decryption of nonce by client	full/partial/none	
		RSA with Cipher Block Chaining	full/partial/none	
6	Snap shots of sample interactions, encryption and decryption: one for each required test input.	Indicate 'full', if your codes can encrypt the following test inputs and then decrypt them correctly, snap shots must be included; 'partial', if your codes can only solve them partially (not all characters can be encrypted, then decrypted back to their original form), snap shots must be included; write 'none', if they are not solvable by your codes.		
	Input containing all letters of the alphabet	Test Input #1: the quick brown fox jumps over the lazy dog	full/partial/none	Must include snap shots
	Input with repeating characters	Test Input #2: AAA	full/partial/none	The encrypted message must not exhibit any repeating pattern as cipher block chaining addresses this problem. Show

				snapshots.
	Input with repeating characters	Test Input #3: 555	full/partial/none	The encrypted message must not exhibit any repeating pattern as cipher block chaining addresses this problem. Show snapshots.
7	Extra work done (Max. bonus of 2 marks)	Program uses the Boost library's big number facility. Very large public key and private keys are being used.		Yes/No
		Include instructions on how to install the big number library you used and the compilation instructions.		
		Write CA's large Public key:		
		Write CA's large Private key:		
		Write Server's large Public key:		
		Write Server's large Private key:		

Nothing follows.