Lab 6: Implement PKI-Based Authentication

In this assignment, a PKI-based protocol for authentication and encryption will be implemented. Suppose a system comprises an application server, its clients, and a centralized certificate authority (CA). The CA in this lab is only a toy one. Real deployment of CAs usually have a hierarchical structure.

Initial Setup

a) **CA** has an identity ID_{ca} and its own public key PK_{ca} and private key SK_{ca}. We assume that CA's public key is publicly accessible and verifiable so that the application server and client can use PK_{ca} securely. In addition to message exchanges, the entities also perform operations such as key generation, signature generation/verification, etc. as explained below.

b) **Application Server** has an identity ID_s and its own public key PK_s and private key SK_s.

c) **Client** has an identity ID_c and its own network address: IP address IP_c and $Port_c$.

Note that there is no information pre-shared among the CA, application server, and client except that the CA's public key PK_{ca} and the identities ID_{ca}, ID_s of the CA and application server are universally known.

Protocol

The flows of messages in the whole protocol is shown below. $RSA(\cdot)$ denotes RSA encryption with the specified public key, $DES(\cdot)$ means DES encryption with the specified DES key, and $Sign(\cdot)$ is RSA signature generation with the specified private key.

Step (1): Before serving the clients, the application server (***S***) firstly registers with the certificate authority (**CA**) to obtain its own public/private keys and certificate. The message is encrypted with **CA**'s public key such that only **CA** can decrypt the message. The message includes a temporary DES key K_{tmp1} for **CA** to encrypt the response message. This is necessary because the response contains the private key for **S**, which is very sensitive.

Step (2): **CA** decrypts the message with its own private key and then generates a public/private key pair along with a certificate for the application server **S**. The certificate $Cert_s$ is an RSA signature over some basic information associated with **S** and **CA**.

Step (3) and (4): Client **C** acquires the public key and certificate of **S**. Messages in these steps are plaintexts since no sensitive information is included in this round.

Step (5): Before providing information for registration, **C** first verify the legitimacy of the received public key PK_s and certificate $Cert_s$. $Cert_s$ is in essence an RSA signature signed with **CA**'s private key, so **C** can verify the signature with **CA**'s public key. If verified, it means that the received public key PK_s is indeed the legitimate one for the application server with identity ID_s. Thus, **C** encrypt its identity and address information with PK_s along with a temporary DES key K_{tmp2}, and sends the message to **S**.

Step (6): Upon receiving the registration request, **S** generates a session key (DES key) for secure communication with **C**.

Step (7) and (8): With the session key, **C** can request service data from **S** securely.

You will implement the above protocol. To this end, you need to use proper crypto libraries (crypto, rsa in Python; openssl in C) in your favor to realize RSA operations, including RSA key generation, encryption, decryption, signature generation, signature verification. Also, you could also reuse your functions for socket communication and DES operations.

Hardcoding values. Please hardcode ID_{ca} to "ID-CA", ID_s to "ID-Server", ID_c to "ID-Client", ***req*** to "memo", ***data*** to "take cis3319 class this morning" in your code. Use your own IP address and port for IP_c and $Port_c$ such as "127.0.0.1" and "5000", respectively. Set $Lifetime_{sess}$ to 60 seconds.

Message flows of the protocol

(1) $\mathbf{S} \rightarrow \mathbf{CA}$: $\mathrm{RSA}_{PK_{ca}}[K_{tmp1}||ID_s||TS_1]$

(2) $\mathbf{CA} \rightarrow \mathbf{S}$: $\mathrm{DES}_{K_{tmp1}}[PK_s||SK_s||Cert_s||ID_s||TS_2]$, where $Cert_s = \mathrm{Sign}_{SK_{ca}}[ID_s||ID_{ca}||PK_s]$

(a). Application Server Registration

(3) $\mathbf{C} \rightarrow \mathbf{S}$: $ID_s||TS_3$

(4) $\mathbf{S} \rightarrow \mathbf{C}$: $PK_s||Cert_s||TS_4$

(5) $\mathbf{C} \rightarrow \mathbf{S}$: $\mathrm{RSA}_{PK_s}[K_{tmp2}||ID_c||IP_c||Port_c||TS_5]$

(6) $\mathbf{S} \rightarrow \mathbf{C}$: $\mathrm{DES}_{K_{tmp2}}[K_{sess}||Lifetime_{sess}||ID_c||TS_6]$

(b). Client Registration

(7) $\mathbf{C} \rightarrow \mathbf{S}$: $\mathrm{DES}_{K_{sess}}[req||TS_7]$

(8) $\mathbf{S} \rightarrow \mathbf{C}$: $\mathrm{DES}_{K_{sess}}[data||TS_8]$

(c). Service Request

Printout:

Your codes are supposed to show the following information on screen for each step when executed.

Step (1): print out the ciphertext on both sides, as well as the generated K_{tmp1} on **S** side, and the received K_{tmp1} on **CA** side.

Step (2): print out the ciphertext on both sides, as well as the generated key pair and the $Cert_s$ on **CA** side, and the received key pair and the $Cert_s$ on **S** side.

Step (3) and Step (4): print out the plaintext on both sides. Check the validation of $Cert_s$.

Step (5): print out the ciphertext on both sides, as well as the generated K_{tmp2} on **C** side, and the received K_{tmp2} on **S** side.

Step (6): print out the ciphertext on both sides, as well as the generated K_{sess} on **S** side, and the received K_{sess} on **C** side.

Step (7): print out the ciphertext on both sides, and the received ***req*** message on **S** side. Check the validation of timestamp.

Step (8): print out the ciphertext on both sides, and the received data message on C side. Check the validation of timestamps.

Please note that all the ciphertext and plaintext printed on the screen should be in hex

except ***req*** and ***data***.

Submission

Submit a lab.zip to Canvas including two parts:

- All the code you have for completing the task.
- Documentation
  - Briefly summary how you design your program, problems you encountered and how you solved them.
  - Tests under different situations (Check the previous slides). First, you should test your program under initial settings. Then change the value of some parameters and record the results. Screen captures are recommended.

Rubric

- Application Server Registration (15)
- Client Registration (15)
- Service Request (10)
- Validation of timestamps and ID (10)
- Documentation (20)
  - Program design(10)
  - Your testing results under different situations(10)
- Demo (30)
- No comments(-5)
- Missing the deliverable (-15)