International Institute of Information Technology Hyderabad

System and Network Security (CS5470)

Assignment 2: Implementation of a digital signature scheme under the client-server model

Deadline: February 18, 2019 (Monday)

Total Marks: 100 [Implementation (Coding + correct results): 75, Vice-voce: 25]

Note:- *It is strongly recommended that no student is allowed to copy programs from others. Hence, if there is any duplicate in the assignment, simply both the parties will be given zero marks without any compromisation. Rest of assignments will not be evaluated further and assignment marks will not be considered towards final grading in the course. No assignment will be taken after deadline. Name your programs as roll_no_assign_2_client.c and roll_no_assign_2_server.c for the client and server. Upload your only client.c and server.c files in a zip/tar file to course portal.*

Description of Problem

Assume that two participants, say A (signer) and B (verifier) agree on the following digital signature scheme. The participant A signs a binary message m of any arbitrary length. The participant B can verify that signature by using the public key of A. The following phases related to this scheme are given below.

Key Generation Phase

1. The signer A chooses a generator g in a multiplicative group G, generates a random number $a \in Z_q^*$, where $Z_q = \{0, 1, 2, \ldots, q-1\}$, $Z_q^* = \{1, 2, \ldots, q-1\}$ and q is large prime so that the discrete logarithm problem (DLP) becomes intractable.
2. After that the signer A computes $y_1 = g^a \pmod q$ and $y_2 = y_1^a \pmod q$.
3. The signer A chooses a hash function H: $\{0,1\}^* \times G \times G \rightarrow Z_q$.
4. Finally, the signer A publishes the public keys as (g, y_1, y_2) and keeps a as the master secret key.

Signature Generation Phase

The signer A executes the following steps:

1. Given a message $m \in \{0,1\}^*$ and secret key a as input, the signer A generates a random number $r \in Z_q^*$.
2. The signer A then calculates

$$\begin{aligned} A &= g^r \pmod q \\ B &= y_1^r \pmod q \end{aligned}$$

$$c = H(A||B||m)$$
$$s = ac + r \pmod{q}$$

where $||$ denotes the concatenation operation.

3. The signature is generated as $\sigma = (c, s)$ on the message m.
4. Finally, the signer A sends the signed message $\langle m, \sigma \rangle$ to a verifier.

Signature Verification Phase

After receiving the signed message from A, B verifies the signature using the following signature verification algorithm. If signature is valid set status as *true* (1); otherwise set status as *false* (0).

1. The verifier B computes the following:

$$A' = g^s y_1^{-c} \pmod{q}$$
$$B' = y_1^s y_2^{-c} \pmod{q}$$

2. The verifier B then accepts the signature $\sigma = (c, s)$ on the message m if the following condition holds:

$$H(A'||B'||m) = c$$

Note that to compute $x^{-c} \pmod{q}$, you can first compute $y = x^{-1} \pmod{q}$ using the extended Euclid's gcd algorithm and then to compute $y^c \pmod{q}$ using the repeated square-and-multiply algorithm.

Protocol Messages to be used in implementation

Opcode	*Message*	*Description*
10	PUBKEY	Public key along with global elements sent to the server by the client
20	SIGNEDMSG	Message along with signature sent from client to server
30	VERSTATUS	Signature verification status by the server

Data structure to be used in implementation

```
#define MAX_SIZE 160
#define MAX_LEN 1024

/* Header of a message */
typedef struct {
    int opcode; /* opcode for a message */
    int s_addr; /* source address */
    int d_addr; /* destination address */
} Hdr;
```

```
/* Signature of a message m */
typedef struct {
    char c[MAX_SIZE];
    long s;
} Signature;

/*A general message */
typedef struct {
    Hdr hdr; /* Header for a message */
    long q; /* A large prime */
    long g; /* A primitive root of q */
    long y1, y2; /* Public values generated by user */
    char plaintext[MAX_LEN]; /* Contains a plaintext message*/
    Signature sign; /* Contains the signature on a plaintext message*/
    int ver_status; /* Successful or unsuccessful result in signature verification */
    int dummy; /*dummy variable is used when necessary */
} Msg;
```

Instructions:

1. Write your client.c program in CLIENT directory. Run the program as: ./a.out 127.0.0.1 (for local host loop back)
2. Write your server.c program in SERVER directory. Run the program as: ./a.out
3. You must use existing hash function code (for example, SHA-1) from open source code.
4. Prime number must be chosen very large (for example > 10000) using the Miller-Robin primality test algorithm.
5. For modular inversion, use the Extended Euclid's GCD algorithm.
6. For modular exponentiation, use the repeated square-and-multiply algorithm.
7. Display your all calculations at both sides A and B.
8. Every student is requested to register in course portal for submissions.