International Institute of Information Technology Hyderabad

System and Network Security (CS5470)

Assignment 2: Implementation of a digital signature scheme using the client-server program

Deadline: February 19, 2018 (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. You are restricted to use only C programming language for implementation.*

Description of Problem

Assume that two participants, say A and B 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

Participant A (client) executes the following steps:

1. Select two primes p and q such that $q \mid (p-1)$ using the Miller-Robin primality test algorithm.
2. Select a random integer g with $1 < g < p-1$, such that $\alpha = g^{(p-1)/q} (\mathrm{mod} p)$ and $\alpha > 1$.
3. Select a private key a, $1 \leq a \leq q-1$.
4. Compute $y = \alpha^a (\mathrm{mod} p)$ using the repeated square-and-multiply algorithm.
5. The public key of A is (p, q, α, y). A sends $PUBKEY(p, q, \alpha, y, H(\cdot))$ to the server (B), where $H(\cdot)$ is a secure hash function (for example, secure hash standard (SHA-1) algorithm).

Signature Generation Phase

A signs the message m as follows:

1. Select a random secret integer k, $1 \leq k \leq q-1$.
2. Compute $r = \alpha^k \pmod{p}$, $e = H(m||r)$, and $s = (ae + k) \bmod q$, where $H(\cdot)$ is a one-way cryptographic hash function.

3. Select two random secret integers u and v, $0 < u < q$ and $0 < v < q$, and compute $r' = r\alpha^{-u}y^{v} \bmod p$.
4. Compute $e' = H(m||r')$ such that $e' = e - v$ and $s' = s - u$.
5. A then sends the signed message $SIGNEDMSG(m, (e', s'))$ to the verifier B.

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. Compute $r^* = \alpha^{s'}.y^{-e'} \pmod p$.
2. Check if $H(m||r^*) = e'$? If so, the signature is valid and set status as *true* (1); otherwise, the the signature is invalid and set status as *false* (0).
3. B sends the message $VERSTATUS(status)$ with signature verification result to A.

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 {
    long e;
    char s[MAX_SIZE];
} Signature;

/*A general message */
typedef struct {
    Hdr hdr; /* Header for a message */
    long p; /* A large prime */
    long q; /* A prime factor of (p-1)*/
```

long g; /* $g = h^{(p-1)/q} \mod p$, with $1 < h < (p-1)$ and $g > 1$ */
long y; /* A public key 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 may 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 Test.
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.