Encrypted Socket Creation - Lab

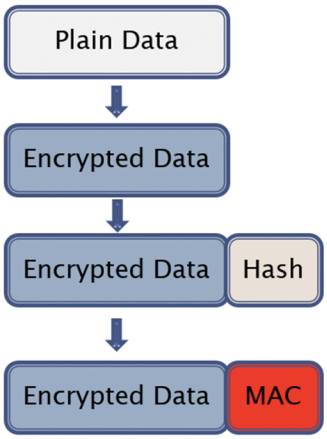
Barak Gonen

Video with Explanation: https://youtu.be/WvOvdBoi2As

In this exercise, we will implement the four mechanisms required to create an encrypted socket:

1. Symmetric key-based encryption known only to both sides
2. Common encryption key selection mechanism
3. Hash function
4. Create a MAC signature by using a public key

This lab examines building these mechanisms. Here is an illustration of the steps:



1. Key-based symmetric encryption

For educational reasons, we use symmetric XOR encryption rather than advanced encryption such as AES. Our encryption key will only be 16 bits. It is not complex to crack this key by brute force, but it illustrates the principle and is not difficult to implement.

On the transmitter side: Any two bytes about to be sent over the socket will first be XORed with the encryption key. If the number of bytes that need to be sent is odd, for simplicity, use only the lower 8 bits of the encryption key.

On the receiving side: Every two bytes received from the socket is XORed with the encryption key. If the number of bytes received is odd, use only the lower 8 bits of the encryption key to decrypt.

2. Common encryption key selection

Determine the key, the shared secret value, using the Diffie-Hellman algorithm. The shared secret should be 16 bits per the encryption method chosen above.

Step A: Select two prime numbers P and G, smaller than 65535 (maximum value of 16 bits)

Step B: Each party selects a private key.

Step C: Each party calculates its own public key and sends it to the other side, using the TCP socket opened between them. At this stage, the information passing through the socket is not yet encrypted and includes only the public keys of both parties.

Step D: Each party calculates the shared secret, a 16-bit number as we consider taking only the remainder of the modulo division by P.

3. **Hash function**

Write a hash function to your liking. The function takes the encrypted message, performs various mathematical operations, and returns a 16-bit hash number.

4. **Public key signature (MAC)**

Use the RSA algorithm to generate the Message Authentication Code (MAC).

Use the following values:

`P=137`

`Q=151`

Party A uses the private key `11669`. Public key `1229`.

Party B uses the private key `7171`. Public Key `2731`.

The signature is generated as follows:

Step A: The sending party calculates the hash of the encrypted message.

Step B: The sending party calculates the value of the signature- the hash to the power of the private key, modulo (P*Q)

You can use the Python `pow` function, for example:

```
signature = pow(hash, 11669, 137*151)
```

Step C: The sending party sends the encrypted message appended with the signature.

Step D: The receiving party uses the sending party's public key to extract the original hash, for example:

```
Received_hash = pow(signature, 1229, 137*151)
```

Step E: The receiving party independently calculates the hash from the rest of the information sent to it.

Step F: The receiving party checks if the calculated hash is the same as `Received_hash` and if not, throws away the received message, as it is not genuine.

Socket Communications

- Stage 1: Establishing the encrypted channel
  - Exchange Diffie-Hellman public keys.
  - Exchange RSA public keys.
- Stage 2: Encrypted communications
  - Messages are encrypted and signed with the MAC.

Assignment

Write a chat between a server and a client (no need to work with multiple clients), messages will be transmitted in encrypted and signed format.