# Experiment: 12

**Title:** Demonstrate various methods of Message Authentication.

**Theory:**

Message authentication is the process of verifying that a message or data has not been tampered with during transmission and that it indeed comes from the claimed source. Various methods of message authentication exist, each with its own strengths and use cases. Here are some commonly used methods:

1. **Message Authentication Code (MAC):**
    - A MAC is a specific code generated using a secret key that is shared between the sender and the recipient.
    - The sender calculates the MAC using a cryptographic algorithm, like HMAC (Hash-based MAC), which combines the message and the secret key.
    - The recipient recalculates the MAC using the received message and the same secret key. If the calculated MAC matches the received MAC, the message is authenticated.
2. **Digital Signatures:**
    - Digital signatures use asymmetric cryptography with a pair of public and private keys.
    - The sender signs the message with their private key to generate a digital signature.
    - The recipient uses the sender's public key to verify the signature. If it's valid, the message is authenticated.
3. **Public Key Infrastructure (PKI):**
    - PKI is a comprehensive system for managing digital keys and certificates.
    - Certificates issued by trusted certificate authorities (CAs) are used to verify the authenticity of messages and entities.
    - It is commonly used for secure web communication, email, and other applications.
4. **Hash Functions:**
    - Hash functions like SHA-256 are used to generate fixed-size hashes from variable-size messages.
    - The sender computes the hash of the message and sends both the message and the hash to the recipient.
    - The recipient calculates the hash of the received message and compares it to the received hash. If they match, the message is considered authentic.
5. **Time Stamping:**

- Time-stamping involves adding a timestamp to a message to prove when it was sent.
- Trusted time-stamping authorities, often referred to as Time Stamp Authorities (TSAs), are responsible for verifying the time of the message.
- Recipients can verify the timestamp to ensure the message hasn't been altered and was sent at the claimed time.

6. **Biometric Authentication:**
    - Biometric methods use unique physical or behavioral characteristics like fingerprints, iris scans, or voice recognition.
    - These methods are used for authenticating individuals rather than messages but can be used to verify the identity of a person sending a message.
7. **Message Authentication without a Key:**
    - This method relies on the inherent structure of the message to detect tampering.
    - For example, appending a checksum or a CRC (Cyclic Redundancy Check) to the message can help verify its integrity without using encryption.
8. **Blockchain Technology:**
    - Blockchain uses distributed ledger technology to provide a tamper-proof and transparent way of verifying the authenticity of data.
    - Once data is added to a blockchain, it's nearly impossible to alter without consensus from the network.
9. **Secure Sockets Layer (SSL) and Transport Layer Security (TLS):**
    - These protocols provide encryption and authentication for data in transit over the internet, ensuring that data exchanged between a client and a server remains confidential and untampered.

## Implementation:

**Digital Signature:**

```
package Exp;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.Signature;
import java.util.Scanner;
public class DigitalSignature {
public static void main(String args[]) throws Exception {
Scanner sc = new Scanner(System.in);
System.out.println("20141224 Pradyumna Bhosale \nEnter some text ");
String msg = sc.nextLine();
KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("DSA");
```

```
keyPairGen.initialize(2048);
KeyPair pair = keyPairGen.generateKeyPair();
PrivateKey privKey = pair.getPrivate();
Signature sign = Signature.getInstance("SHA256withDSA");
sign.initSign(privKey);
byte[] bytes = "msg".getBytes();
sign.update(bytes);
byte[] signature = sign.sign();
System.out.println("\nDigital signature for given text: "+new
String(signature, "UTF8"));
System.out.println("Digital Signature is Verified");
}
}
```

```
20141224 Pradyumna Bhosale
Enter some text
pradyumna

Digital signature for given text: 0>0↔????_hpu*????/?tn??}M@?N?0↔????|?5??QL→?????>¶?%??w
Digital Signature is Verified
```

**Hash Functions:**

```
package Exp;
import java.math.BigInteger;
import java.security.*;
public class ExpSHA1 {
public static String encryptThisString(String input)
{
try {
MessageDigest md = MessageDigest.getInstance("SHA-1");
byte[] messageDigest = md.digest(input.getBytes());
BigInteger no = new BigInteger(1, messageDigest);
String hashtext = no.toString(16);
while (hashtext.length() < 32) {
hashtext = "0" + hashtext;
}
return hashtext;
}
catch (NoSuchAlgorithmException e) {
throw new RuntimeException(e);
}
}
```

```
public static void main(String args[]) throws
NoSuchAlgorithmException
{
System.out.println("Pradyumna Bhosale 20141224");
System.out.println("HashCode Generated by SHA-1 is: ");
String s1 = "Pradyumna";
System.out.println("\n" + s1 + " : " + encryptThisString(s1));
}
}
```

**Output:**

```
Pradyumna Bhosale 20141224
HashCode Generated by SHA-1 is:

Pradyumna : 523040ed4681a1e425db5388e37ac74d679c23ab
PS D:\prog\Java practise>
```

**HMAC:**

```
package Exp12;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.security.NoSuchAlgorithmException;
import java.security.InvalidKeyException;
import java.util.Base64;

public class HMACExample {

    public static void main(String[] args) {
        try {
            // Your secret key (should be kept secret)
            String secretKey = "MySecretKey";
            String message = "Hello, HMAC!";

            // Create a SecretKeySpec object from the secret key
            SecretKeySpec secretKeySpec = new
SecretKeySpec(secretKey.getBytes(), "HmacSHA256");

            // Initialize the MAC with the HmacSHA256 algorithm and the secret
key
            Mac mac = Mac.getInstance("HmacSHA256");
            mac.init(secretKeySpec);

            // Calculate the HMAC
            byte[] hmac = mac.doFinal(message.getBytes());
```

```
            // Encode the HMAC as a Base64 string for easy storage and
transmission
            String encodedHmac = Base64.getEncoder().encodeToString(hmac);

            System.out.println("Message: " + message);
            System.out.println("HMAC: " + encodedHmac);

            // To verify the HMAC, you can repeat the process with the
received HMAC and compare it to the calculated HMAC.
        } catch (NoSuchAlgorithmException | InvalidKeyException e) {
            e.printStackTrace();
        }
    }

}
```

```
Message: Hello, HMAC!
HMAC: k8alqpaLEZO8fYRjEoKRWvM0hRiadIQxKQ30eRakMAY=
PS D:\prog\Java practise>
```