MIT-WPU

॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

SY B.Tech Semester-IV (AY 2022-23)

Computer Science and Engineering (Cybersecurity and Forensics)

Assign No.	List of Assignments
1.	Write a program using JAVA or Python or C++ to implement any classical cryptographic technique.
2.	Write a program using JAVA or Python or C++ to implement Feistal Cipher structure
3.	Write a program using JAVA or Python or C++ to implement S-AES symmetric key algorithm.
4.	Write a program using JAVA or Python or C++ to implement RSA asymmetric key algorithm.
5.	Write a program using JAVA or Python or C++ to implement integrity of message using MD5 or SHA
6.	Write a program using JAVA or Python or C++ to implement Diffie Hellman Key Exchange Algorithm
7.	**Write a program using JAVA or Python or C++ to implement Digital signature using DSA.**
8.	Demonstrate Email Security using - PGP or S/MIME for Confidentiality, Authenticity and Integrity.
9.	Demonstration of secured web applications system using SSL certificates and its deployment in Apache tomcat server
10.	Configuration and demonstration of Intrusion Detection System using Snort.
11.	Configuration and demonstration of NESSUS tool for vulnerability assessment.

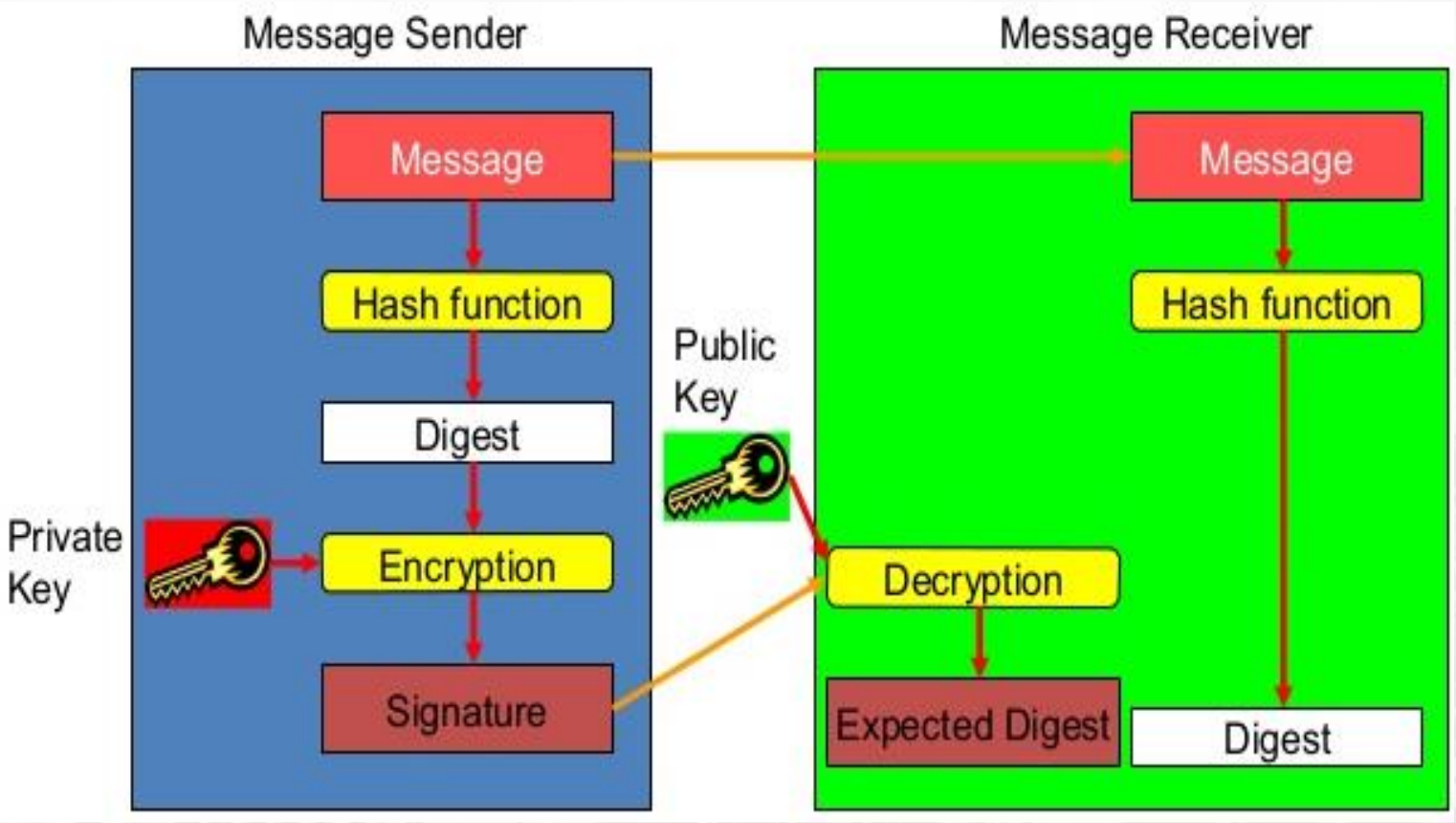
Laboratory: Lab Assignment

7	Write a program using JAVA or Python or C++ to implement Digital signature using DSA.

Digital Signature Techniques

- Digital Signature Standard (DSS) uses SHA-1
- RSA and DSA

RSA and Digital Signature



Digital Signature Algorithm (DSA)

- creates a 320 bit signature
- smaller and faster than RSA
- a digital signature scheme only
- security depends on difficulty of computing discrete logarithms
- variant of ElGamal & Schnorr schemes

DSA Key Generation

- have shared **global public key** values (**p, q, g**):
  - choose a large prime **p** with $2^{L-1} < p < 2^L$
    - where L= 512 to 1024 bits and is a multiple of 64
  - choose 160-bit prime number q
    - such that q is a 160 bit prime divisor of (p-1)
  - choose $g = h^{(p-1)/q}$
    - where $1 < h < p-1$ and $h^{(p-1)/q} \bmod p > 1$
- users choose **private key** & compute **public key**:
  - choose random private key: $x < q$
  - compute public key: $y = g^x \bmod p$

Note: L will be one member of the set {512, 576, 640, 704, 768, 832, 896, 960, 1024}

DSA Signature Creation

- to **sign** a message M the sender:
  - generates a random signature key k, k < q
- then computes signature pair:

$$\mathbf{r = (g^k \bmod p) \bmod q}$$

$$\mathbf{s = [k^{-1}(SHA(M) + x * r)] \bmod q}$$

- sends **signature (r, s)** with message M

DSA Signature Verification

- having received M & signature (r, s)
- to **verify** a signature, recipient computes:

$w = s^{-1} \bmod q$

$u_1 = [SHA(M) * w] \bmod q$

$u_2 = (r * w) \bmod q$

$v = [(g^{u1} * y^{u2}) \bmod p] \bmod q$

- if **v = r** then signature is verified

Java Package

1. First, import the required Java security packages and Base64 encoding.
2. Next, define a main method in which we generate a DSA key pair.
3. Create a string message to sign and convert it to bytes.
4. Create a Signature object with the getInstance method and pass it the string "SHA1withDSA" as the algorithm parameter. This specifies the hash algorithm to use for the signature.
5. Initialize the Signature object with the private key of the key pair using the initSign method.
6. Update the Signature object with the message to be signed using the update method.

7. Generate the signature using the sign method and store it in a byte array.

8. Convert the signature byte array to Base64 encoding and print it out to the console.

9. To verify the signature, create a new Signature object and initialize it with the public key of the key pair using the initVerify method.

10. Update the Signature object with the message to be verified using the update method.

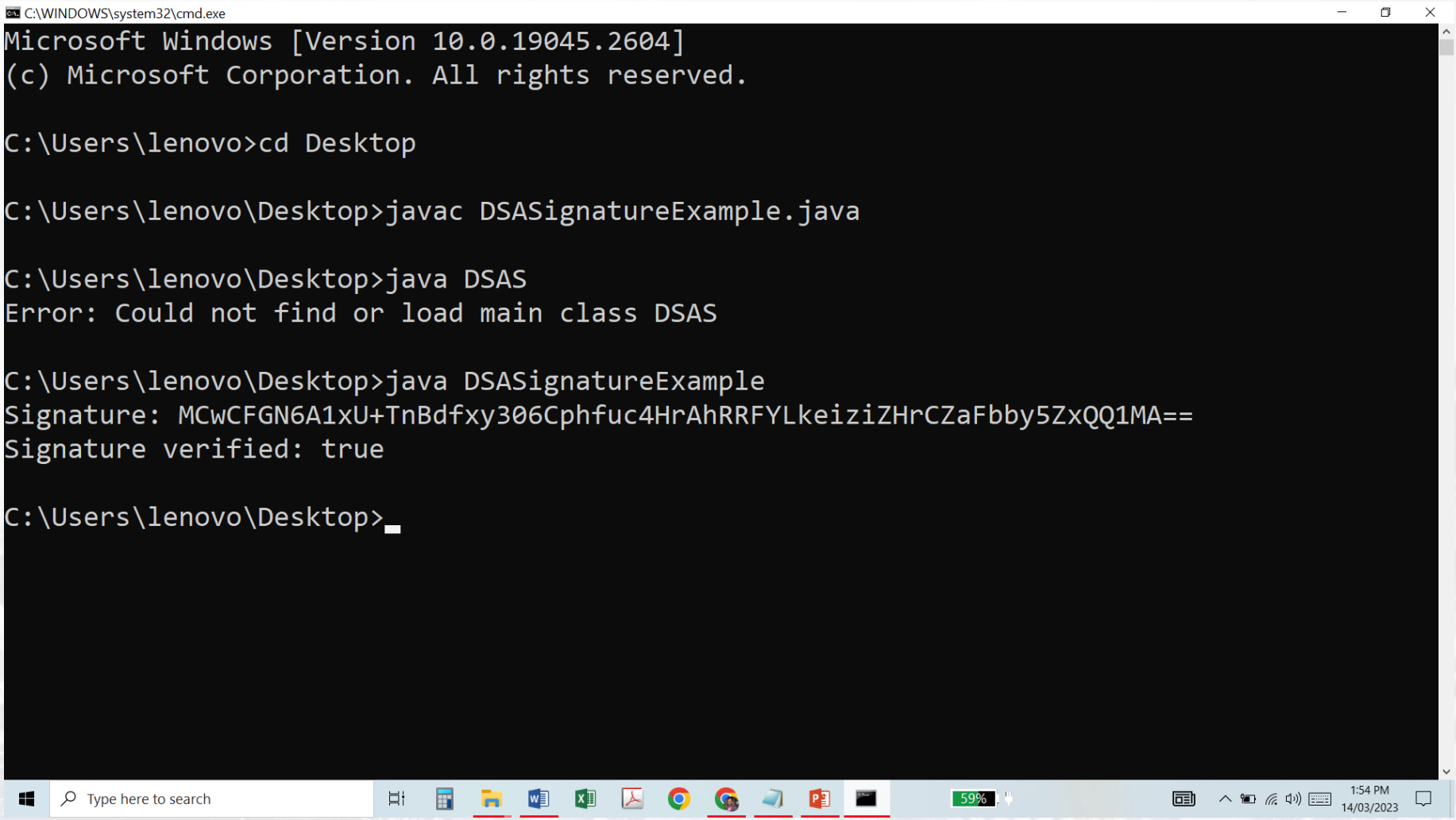
11. Call the verify method on the Signature object to verify the signature using the public key.

12. Display the result of the signature verification to the console.

```
import java.security.*;
import java.util.Base64;

public class DSASignatureExample {

    public static void main(String[] args) throws Exception {

        // Generate DSA key pair
        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA");
        keyGen.initialize(1024);
        KeyPair keyPair = keyGen.generateKeyPair();
        PrivateKey privateKey = keyPair.getPrivate();
        PublicKey publicKey = keyPair.getPublic();

        // Create a message to sign
        String message = "This is a message to be signed.";
        byte[] messageBytes = message.getBytes("UTF-8");
```

```
// Create a signature object
        Signature dsa = Signature.getInstance("SHA1withDSA");

        // Initialize the signature object with the private key
        dsa.initSign(privateKey);

        // Update the signature object with the message to be signed
        dsa.update(messageBytes);

        // Generate the signature
        byte[] signatureBytes = dsa.sign();

        // Print out the signature in Base64 encoding
        String signatureBase64 = Base64.getEncoder().encodeToString(signatureBytes);
        System.out.println("Signature: " + signatureBase64);
```

```
// Verify the signature using the public key
        Signature dsaVerify = Signature.getInstance("SHA1withDSA");
        dsaVerify.initVerify(publicKey);
        dsaVerify.update(messageBytes);

        boolean signatureVerified = dsaVerify.verify(signatureBytes);
        System.out.println("Signature verified: " + signatureVerified);
    }
}
```


C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.19045.2604]
(c) Microsoft Corporation. All rights reserved.
C:\Users\lenovo>cd Desktop
C:\Users\lenovo\Desktop>javac DSASignatureExample.java
C:\Users\lenovo\Desktop>java DSAS
Error: Could not find or load main class DSAS
C:\Users\lenovo\Desktop>java DSASignatureExample
Signature: MCwCFGN6A1xU+TnBdfxy306Cphfuc4HrAhRRFYLkeiziZHrCZaFbby5ZxQQ1MA==
Signature verified: true
C:\Users\lenovo\Desktop>
Type here to search
59%
1:54 PM
14/03/2023