Name: Parshwa Herwade

PRN: 22510064

BTech Final Year CSE

Cryptography and Network Security Lab (B – 1)

Assignment 10 (Implement the SIGNATURE SCHEME – Digital Signature Standard)

**Objectives:**

To implement the **Digital Signature Standard (DSS)** using the **Digital Signature Algorithm (DSA)** to create and verify digital signatures, thereby ensuring the authenticity and integrity of electronic messages or documents.

**Problem Statement:**In digital communication, ensuring the authenticity of a sender and the integrity of the message is critical. A digital signature serves as a cryptographic technique to validate both.

Design and implement a system using the Digital Signature Standard (DSS) that performs the following tasks:

1. Key Generation:
   * Generate a private key (used for signing) and a public key (used for verifying).
   * Use parameters such as large prime numbers and hash functions (e.g., SHA-1 or SHA-256) as required by DSA.
2. Message Signing:
   * Accept a message (text input).
   * Use the private key to generate a digital signature for the message using the DSA algorithm.
3. Signature Verification:
   * Use the corresponding public key to verify the digital signature against the original message.
   * Confirm whether the signature is valid or has been tampered with.
4. Demonstrate Security:
   * Show that any alteration in the message or signature results in a failed verification.
   * Explain how this ensures message integrity, authentication, and non-repudiation.

The system should simulate a real-world use case where one party signs a message and another party verifies it, highlighting how digital signatures help in secure digital communications, such as email signing, legal document authentication, or secure software distribution.

1. import java.nio.charset.StandardCharsets;

2. import java.security.\*;

3. import java.security.spec.\*;

4. import java.util.Base64;

5. import java.util.Scanner;

6. import java.util.logging.Level;

7. import java.util.logging.Logger;

8.

9. public class DSASignature {

10.     private static final Logger logger = Logger.getLogger(DSASignature.class.getName());

11.

12.     private KeyPair keyPair;

13.

14.     public DSASignature() {

15.         try {

16.             generateKeys(2048);

17.         } catch (Exception e) {

18.             logger.log(Level.SEVERE, "Failed to generate keys on startup", e);

19.         }

20.     }

21.

22.     public void generateKeys(int keySize) throws NoSuchAlgorithmException {

23.         KeyPairGenerator kpg = KeyPairGenerator.getInstance("DSA");

24.         SecureRandom random = SecureRandom.getInstanceStrong();

25.         kpg.initialize(keySize, random);

26.         this.keyPair = kpg.generateKeyPair();

27.         logger.info("DSA KeyPair generated (keySize=" + keySize + ")");

28.     }

29.

30.     public String getPublicKeyBase64() {

31.         if (keyPair == null) return null;

32.         byte[] encoded = keyPair.getPublic().getEncoded();

33.         return Base64.getEncoder().encodeToString(encoded);

34.     }

35.

36.     public String getPrivateKeyBase64() {

37.         if (keyPair == null) return null;

38.         byte[] encoded = keyPair.getPrivate().getEncoded();

39.         return Base64.getEncoder().encodeToString(encoded);

40.     }

41.

42.     public String signMessage(String message) {

43.         try {

44.             Signature signer = Signature.getInstance("SHA256withDSA");

45.             signer.initSign(keyPair.getPrivate());

46.             signer.update(message.getBytes(StandardCharsets.UTF\_8));

47.             byte[] sigBytes = signer.sign();

48.             return Base64.getEncoder().encodeToString(sigBytes);

49.         } catch (Exception e) {

50.             logger.log(Level.SEVERE, "Signing failed", e);

51.             return null;

52.         }

53.     }

54.

55.     public boolean verifySignature(String message, String base64Signature) {

56.         try {

57.             Signature verifier = Signature.getInstance("SHA256withDSA");

58.             verifier.initVerify(keyPair.getPublic());

59.             verifier.update(message.getBytes(StandardCharsets.UTF\_8));

60.             byte[] sigBytes = Base64.getDecoder().decode(base64Signature);

61.             return verifier.verify(sigBytes);

62.         } catch (Exception e) {

63.             logger.log(Level.WARNING, "Verification failed", e);

64.             return false;

65.         }

66.     }

67.

68.     public PublicKey importPublicKeyFromBase64(String base64PublicKey) throws GeneralSecurityException {

69.         byte[] keyBytes = Base64.getDecoder().decode(base64PublicKey);

70.         X509EncodedKeySpec spec = new X509EncodedKeySpec(keyBytes);

71.         KeyFactory kf = KeyFactory.getInstance("DSA");

72.         return kf.generatePublic(spec);

73.     }

74.

75.     public boolean verifyWithPublicKey(String base64PublicKey, String message, String base64Signature) {

76.         try {

77.             PublicKey pub = importPublicKeyFromBase64(base64PublicKey);

78.             Signature verifier = Signature.getInstance("SHA256withDSA");

79.             verifier.initVerify(pub);

80.             verifier.update(message.getBytes(StandardCharsets.UTF\_8));

81.             byte[] sigBytes = Base64.getDecoder().decode(base64Signature);

82.             return verifier.verify(sigBytes);

83.         } catch (Exception e) {

84.             logger.log(Level.WARNING, "Verification with provided public key failed", e);

85.             return false;

86.         }

87.     }

88.

89.     public static void main(String[] args) {

90.         Scanner sc = new Scanner(System.in);

91.         DSASignature demo = new DSASignature();

92.

93.         System.out.println("=== DSS (DSA) Digital Signature Demo ===");

94.         while (true) {

95.             System.out.println("\nMenu:");

96.             System.out.println("1. Generate new DSA key pair (2048 bits)");

97.             System.out.println("2. Show public key (Base64)");

98.             System.out.println("3. Show private key (Base64) [FOR DEMO ONLY]");

99.             System.out.println("4. Sign a message");

100.             System.out.println("5. Verify a signature (using stored public key)");

101.             System.out.println("6. Verify signature with provided public key (Base64)");

102.             System.out.println("7. Demonstrate tamper detection");

103.             System.out.println("8. Exit");

104.             System.out.print("Enter choice: ");

105.

106.             String line = sc.nextLine().trim();

107.             int choice;

108.             try {

109.                 choice = Integer.parseInt(line);

110.             } catch (NumberFormatException nfe) {

111.                 System.out.println("Invalid input. Enter a number 1-8.");

112.                 continue;

113.             }

114.

115.             try {

116.                 switch (choice) {

117.                     case 1:

118.                         demo.generateKeys(2048);

119.                         System.out.println("New DSA key pair generated (2048 bits).");

120.                         break;

121.                     case 2:

122.                         System.out.println("Public Key (Base64, X.509):");

123.                         System.out.println(demo.getPublicKeyBase64());

124.                         break;

125.                     case 3:

126.                         System.out.println("Private Key (Base64, PKCS#8) — DEMO ONLY:");

127.                         System.out.println(demo.getPrivateKeyBase64());

128.                         break;

129.                     case 4:

130.                         System.out.print("Enter message to sign: ");

131.                         String msgToSign = sc.nextLine();

132.                         String signature = demo.signMessage(msgToSign);

133.                         System.out.println("\nSignature (Base64):");

134.                         System.out.println(signature);

135.                         break;

136.                     case 5:

137.                         System.out.print("Enter message to verify: ");

138.                         String msgToVerify = sc.nextLine();

139.                         System.out.print("Enter signature (Base64): ");

140.                         String sigInput = sc.nextLine();

141.                         boolean ok = demo.verifySignature(msgToVerify, sigInput);

142.                         System.out.println("Verification result: " + (ok ? "VALID" : "INVALID"));

143.                         break;

144.                     case 6:

145.                         System.out.print("Enter public key (Base64, X.509): ");

146.                         String pubKeyInput = sc.nextLine();

147.                         System.out.print("Enter message to verify: ");

148.                         String msgPk = sc.nextLine();

149.                         System.out.print("Enter signature (Base64): ");

150.                         String sigPk = sc.nextLine();

151.                         boolean ok2 = demo.verifyWithPublicKey(pubKeyInput, msgPk, sigPk);

152.                         System.out.println("Verification with provided public key: " + (ok2 ? "VALID" : "INVALID"));

153.                         break;

154.                     case 7:

155.                         System.out.print("Enter message to sign for tamper demo: ");

156.                         String original = sc.nextLine();

157.                         String sig = demo.signMessage(original);

158.                         System.out.println("\nOriginal signature (Base64): " + sig);

159.

160.                         String tamperedMessage = original + "X";

161.                         boolean v1 = demo.verifySignature(original, sig);

162.                         boolean v2 = demo.verifySignature(tamperedMessage, sig);

163.                         System.out.println("\nVerification with original message: " + (v1 ? "VALID" : "INVALID"));

164.                         System.out.println("Verification with tampered message:  " + (v2 ? "VALID" : "INVALID"));

165.                         System.out.println("\nConclusion: tampering the message invalidates the signature.");

166.                         break;

167.                     case 8:

168.                         System.out.println("Exiting. Goodbye!");

169.                         sc.close();

170.                         return;

171.                     default:

172.                         System.out.println("Enter a valid choice (1-8).");

173.                 }

174.             } catch (Exception e) {

175.                 logger.log(Level.SEVERE, "Unexpected error in menu operation", e);

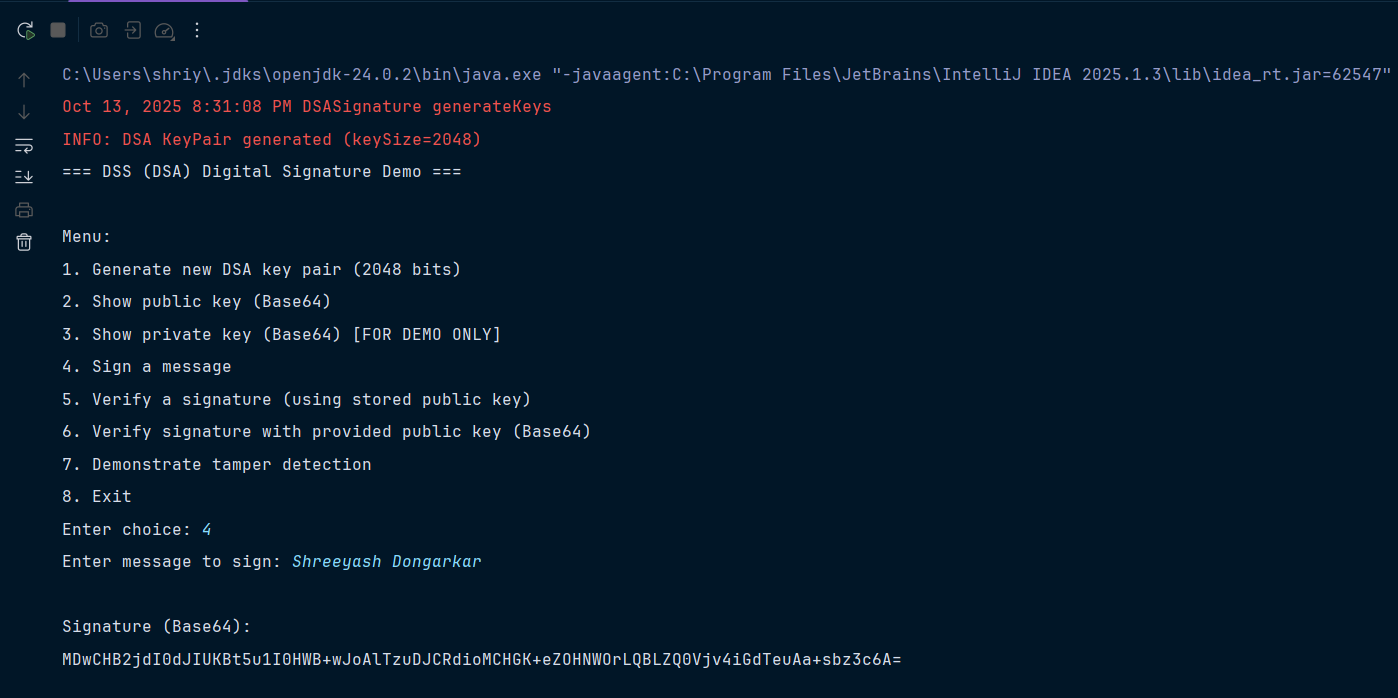
176.             }

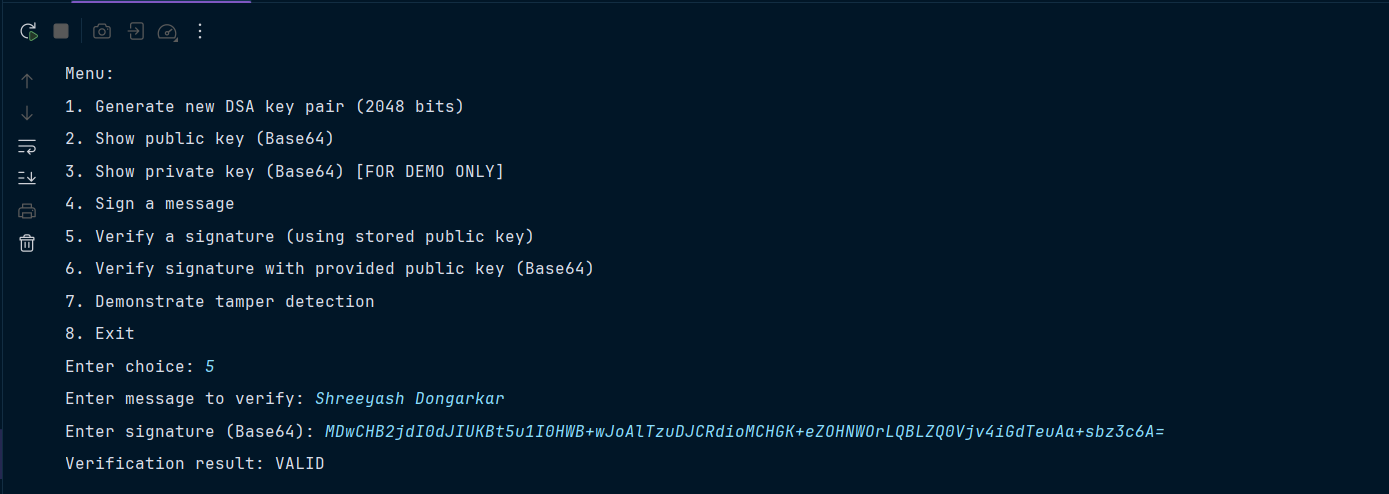
177.         }

178.     }

179. }

180.







**Equipment / Tools:**

* Java JDK 8+ (JDK 11 or later recommended)
* Any Java IDE or text editor (IntelliJ, Eclipse, VS Code)
* Command-line terminal for compile/run
* No external libraries required — uses standard java.security and java.util packages

**Theory:**

The **Digital Signature Algorithm (DSA)** is part of the DSS. It signs a hash of the message (not the raw message).

Typical signature flow:

1. Choose cryptographic hash (SHA-256 recommended).
2. Signer computes hash of message, then signs hash with private DSA key producing a signature (r,s).
3. Verifier computes hash of received message and uses signer’s public DSA key to verify signature.

Properties of digital signatures:

* **Integrity:** Any change in message modifies the hash and makes verification fail.
* **Authentication:** Signature proves the holder of private key signed the message.
* **Non-repudiation:** Signer cannot later deny the signature (if private key is kept secure).

**Procedure:**

1. Generate a DSA KeyPair (private + public) using KeyPairGenerator.  
2. Accept message text from user.  
3. Use Signature object initialized with the private key and algorithm "SHA256withDSA" to sign message bytes.  
4. Output signature in Base64.  
5. To verify, initialize Signature with public key and algorithm "SHA256withDSA", feed the message bytes and call verify() with the Base64-decoded signature.  
6. Demonstrate tampering by changing the message or the signature and showing verification fails.

**Steps:**

1. Start program.
2. Choose “Generate keys” (or let program generate on startup).
3. Choose “Sign message”, enter message — program prints Base64 signature.
4. Choose “Verify signature”, paste message and signature — program prints verification result.
5. Try changing a character in the message (or signature) and verify again — it fails.

**Observations:**

* Valid signature verifies successfully only when the exact original message and the valid signature are used.
* Any change (single bit or character) in the message produces a different hash and verification fails.
* Signature encoded in Base64 is compact and safe for copy/paste/storage.

**Conclusion:**

* DSA (under DSS) with a secure hash function provides strong guarantees of message **integrity** and **authenticity** when keys are kept secure.
* Practical considerations: use secure key storage, protect private keys, choose appropriate key sizes and hash algorithms (DSA+SHA-256, key ≥ 2048 bits).
* Digital signatures are widely used in email signing, document signing, secure code distribution, and many authentication protocols.