Crypto

Des

```
#include <stdio.h>
#include <stdint.h>

uint32_t left_rotate(uint32_t value, int shifts, int bits) {
  return ((value << shifts) | (value >> (bits - shifts))) & ((1 << bits) - 1);
}

void generate_subkeys(uint64_t initial_key, uint64_t subkeys[16]) {
  int shift_schedule[16] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};

  uint32_t left_half = (initial_key >> 36) & 0x0FFFFFFF;
  uint32_t right_half = (initial_key >> 8) & 0x0FFFFFFF;

  for (int i = 0; i < 16; i++) {
    left_half = left_rotate(left_half, shift_schedule[i], 28);
    right_half = left_rotate(right_half, shift_schedule[i], 28);
    subkeys[i] = ((uint64_t)left_half << 28) | right_half;
  }
}

int main() {
  uint64_t initial_key = 0x133457799BBCDFF1;
  uint64_t subkeys[16];

  generate_subkeys(initial_key, subkeys);

  for (int i = 0; i < 16; i++) {
    printf("Subkey %2d: %012lx\n", i + 1, subkeys[i] & 0xFFFFFFFFFFFF);
  }
```

```c
    return 0;
}
```

....

```c
#include <stdio.h>

void permute(int *input, int *output, int *perm, int size) {
    for (int i = 0; i < size; i++) {
        output[i] = input[perm[i] - 1];
    }
}

void leftShift(int *key, int shifts) {
    int temp[10];
    for (int i = 0; i < 10; i++) {
        temp[i] = key[(i + shifts) % 10];
    }
    for (int i = 0; i < 10; i++) {
        key[i] = temp[i];
    }
}

void generateKeys(int *key, int *k1, int *k2) {
    int p10[10] = {3, 5, 2, 7, 4, 10, 1, 9, 8, 6};
    int p8[8] = {6, 7, 8, 5, 4, 3, 2, 1};

    int temp[10];
    permute(key, temp, p10, 10);
    leftShift(temp, 1);
    permute(temp, k1, p8, 8);

    leftShift(temp, 2);
    permute(temp, k2, p8, 8);
}

int main() {
    int key[10];
    int k1[8], k2[8];

    printf("Enter a 10-bit key (binary digits only): ");
    for (int i = 0; i < 10; i++) {
        scanf("%1d", &key[i]);
    }

    generateKeys(key, k1, k2);

    printf("Key K1: ");
    for (int i = 0; i < 8; i++) printf("%d", k1[i]);

    printf("\nKey K2: ");
    for (int i = 0; i < 8; i++) printf("%d", k2[i]);

    printf("\n"); // To add a newline at the end
    return 0;
}
```

SHA 512

```
import java.security.MessageDigest;
```

```
}


import java.security.NoSuchAlgorithmException;
import java.util.Scanner;

public class SHA512 {
  public static void main(String[] args) {
    Scanner inputScanner = new Scanner(System.in);
    System.out.print("Enter input: ");
    String userInput = inputScanner.nextLine();
    String sha512Hash = generateSHA512Hash(userInput);
    System.out.println("SHA-512 hash of \"" + userInput + "\": " + sha512Hash);
    inputScanner.close();
  }

  public static String generateSHA512Hash(String userInput) {
    try {
      MessageDigest sha512Digest = MessageDigest.getInstance("SHA-512");
      byte[] hashBytes = sha512Digest.digest(userInput.getBytes());
      StringBuilder hexString = new StringBuilder();
      for (byte b : hashBytes) {
        String hex = Integer.toHexString(0xff & b);
        if (hex.length() == 1) hexString.append('0');
        hexString.append(hex);
      }
      return hexString.toString();
    } catch (NoSuchAlgorithmException e) {
      throw new RuntimeException(e);
    }
  }
}
```

Digital Signature

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;
import java.util.Scanner;

public class DigitalSignatureGenerator {
    public static void main(String[] args) {
        try {
            Scanner userInputScanner = new Scanner(System.in);
            System.out.print("Enter input: ");
            String userMessage = userInputScanner.nextLine();

            KeyPairGenerator keyGenerator = KeyPairGenerator.getInstance("DSA");
            keyGenerator.initialize(1024);
            KeyPair keyPair = keyGenerator.generateKeyPair();
            PrivateKey privateKey = keyPair.getPrivate();
            PublicKey publicKey = keyPair.getPublic();

            byte[] digitalSignature = generateSignature(userMessage, privateKey);
            System.out.println("Digital Signature: " + bytesToHexadecimal(digitalSignature));

            boolean isSignatureVerified = verifyDigitalSignature(userMessage, digitalSignature, publicKey);
            System.out.println("Signature Verified: " + isSignatureVerified);

            userInputScanner.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static byte[] generateSignature(String data, PrivateKey privateKey) throws Exception {
        Signature signatureGenerator = Signature.getInstance("SHA1withDSA");
        signatureGenerator.initSign(privateKey);
        signatureGenerator.update(data.getBytes());
        return signatureGenerator.sign();
    }

    public static boolean verifyDigitalSignature(String data, byte[] signature, PublicKey publicKey) throws Exception {
        Signature signatureVerifier = Signature.getInstance("SHA1withDSA");
        signatureVerifier.initVerify(publicKey);
        signatureVerifier.update(data.getBytes());
        return signatureVerifier.verify(signature);
    }

    public static String bytesToHexadecimal(byte[] bytes) {
        StringBuilder hexadecimalString = new StringBuilder();
        for (byte b : bytes) {
            String hexadecimal = Integer.toHexString(0xff & b);
            if (hexadecimal.length() == 1) hexadecimalString.append('0');
            hexadecimalString.append(hexadecimal);
        }
        return hexadecimalString.toString();
    }
}
```