

Department of Information and Communication Technology

Faculty of Technology

University of Ruhuna

**Assignment 01**

**Lab sheet 01 (Tasks 6)**

**Network, Computer and Application Security**

**ICT-** **3243**

**Submitted by:**

TG/2020/675

S.K.M Perera

**Submitted to:**

Lecturer in Charge: (Probationary) Ms. RDN Shakya

**Date of submission:**

1st November 2024

RandomShift AES Encryption and Decryption Program

**Design a logic**

**01). generateKey() function**

**Purpose:** generate a 256bit AES encryption key

**How it works:** using AES algorithm create a AES encryption key use for encryption and decryption

**02). keyToString() function**

**Purpose:** convert secret objects to string for easier storage and shearing.

**How it works:** using base64 encoding covert the byte array of the key

**Step 03). StringToKey() function**

**Purpose:** convert a string format of a key back to secret key

**How it works:** This function takes the Base64-encoded key and returns bytes. Then, reconstructs the SecretKey using the bytes obtained with a new SecretKeySpec that specifies the AES algorithm.

**04.) encrypt() function**

**Purpose:** Encodes a message with AES and applies some random character shift for extra obscurity.

**How it works:**

* Creates an AES Cipher instance with encryption, using the provided SecretKey.
* Encrypts the body of message bytes and encodes to Base64
* Creates a random shift (1-10) and then shifts only characters of the encrypted string with this value(shiftString).
* Here it returns the shift value and shifted encrypted message in a single string separated by a colon, to make sure decrypt method knows how much offset was done.

1. **Decrypt() function**

**Purpose**: For decrypting the message by inverting the random shift and employing AES encryption algorithm.

**How it works:**

* Divides the input string into the shift and shifted messages with the use of split(“:”)
* Determines a new shift in order to undo character shift by making use of negative of the shiftString.
* Recovers bytes from the Base64 encoded reversed string: through this step in essence it is all RSIW position already.
* With the provided SecretKey, these bytes are decrypted with the AES Cipher in decryption mode.
* Reestablishes the potentially altered bytes having been decrypted back to a string that gives out the original message.

1. **shiftString () function**

**Purpose:** Shifts positions of each character in the string lines up to a given number of positions of that character.

**How it works:** For each character position in the input string, the shift amount is added (or subtracted during decryption) to that character’s ASCII code, and the result is kept in a StringBuilder. As a result, a shifted string is produced.

1. **Main Class**

**Purpose**: Lets the user interact with the program in a way that they are able to encrypt messages or decrypt messages or exit.

**How it works**

* Employs a loop that repeatedly asks the user to take an action which is either Encrypt a message, Decrypt a message or Exit the program.
* In case He chooses Encrypt:
* Invokes generateKey and creates an encryption key which is then converted to string format using keyToString while asking the user for a message.
* Allows for the message to be compressed using encrypt and provides the recipient with the concealed message and the encryption key.
* In case He chooses Decrypt:
* After an encrypted message and a key string is entered, the system converts the key string to SecretKey using stringToKey and tries to decrypt using decrypt.
* Provided that this succeeds, the user is shown the message that was decrypted, in the other case an error is displayed that decryption attempts had been unsuccessful.
* In case He chooses Exit: Closes the application.

**A diagram of a flowchart

Description automatically generatedFlow Chart**

**Pseudo Code**

*Pseudo Logic for Encryption Process*

**FUNCTION generateKey()**

**// Generate a new AES key of 256 bits**

**INITIALIZE KeyGenerator for "AES"**

**SET keyGen to KeyGenerator instance**

**INITIALIZE keyGen with 256 bits**

**RETURN keyGen.generateKey()**

**FUNCTION keyToString(secretKey)**

**// Convert the SecretKey to a Base64 encoded string**

**RETURN Base64.encode(secretKey.encoded)**

**FUNCTION stringToKey(keyStr)**

**// Decode the Base64 string back to bytes**

**SET decodedKey to Base64.decode(keyStr)**

**RETURN new SecretKeySpec(decodedKey, "AES")**

**FUNCTION encrypt(message, secretKey)**

**// Initialize AES cipher in encrypt mode**

**INITIALIZE cipher with "AES"**

**cipher.init(ENCRYPT\_MODE, secretKey)**

**// Encrypt the message**

**SET encryptedBytes to cipher.doFinal(message.bytes)**

**SET encryptedMessage to Base64.encode(encryptedBytes)**

**// Generate a random shift value between 1 and 10**

**SET shift to random integer between 1 and 10**

**SET shiftedMessage to shiftString(encryptedMessage, shift)**

**// Return shift value and shifted message**

**RETURN shift + ":" + shiftedMessage**

**FUNCTION decrypt(shiftedMessageWithKey, secretKey)**

**// Split the input into shift value and shifted message**

**SET parts to split(shiftedMessageWithKey, ":")**

**SET shift to integer(parts[0])**

**SET shiftedMessage to parts[1]**

**// Reverse the character shift**

**SET encryptedMessage to shiftString(shiftedMessage, -shift)**

**// Initialize AES cipher in decrypt mode**

**INITIALIZE cipher with "AES"**

**cipher.init(DECRYPT\_MODE, secretKey)**

**// Decrypt the message**

**SET decodedBytes to Base64.decode(encryptedMessage)**

**SET decryptedBytes to cipher.doFinal(decodedBytes)**

**RETURN decryptedBytes as string**

**FUNCTION shiftString(input, shift)**

**// Shift each character in the input string**

**INITIALIZE shifted to empty string**

**FOR each character c in input DO**

**APPEND (c + shift) to shifted**

**RETURN shifted**

**FUNCTION main()**

**INITIALIZE scanner for console input**

**SET continueProgram to true**

**WHILE continueProgram DO**

**PRINT "Would you like to (1) Encrypt, (2) Decrypt, or (3) Exit?"**

**READ choice**

**SWITCH choice DO**

**CASE 1: // Encryption flow**

**SET secretKey to generateKey()**

**SET keyString to keyToString(secretKey)**

**PRINT "Enter a message to encrypt:"**

**READ message**

**SET encryptedMessage to encrypt(message, secretKey)**

**PRINT "Encrypted Message: " + encryptedMessage**

**PRINT "Encryption Key (save this securely): " + keyString**

**CASE 2: // Decryption flow**

**PRINT "Enter the encrypted message to decrypt:"**

**READ encryptedInput**

**PRINT "Enter the key for decryption:"**

**READ keyInput**

**TRY**

**SET userKey to stringToKey(keyInput)**

**SET decryptedMessage to decrypt(encryptedInput, userKey)**

**PRINT "Decrypted Message: " + decryptedMessage**

**CATCH Exception**

**PRINT "Decryption failed: Incorrect key or message format."**

**CASE 3: // Exit**

**SET continueProgram to false**

**PRINT "Exiting the program. Goodbye!"**

**DEFAULT:**

**PRINT "Invalid choice. Please select 1 for Encryption, 2 for Decryption, or 3 to Exit."**

**Verify the logic**

**Encryption Process**

**Step 01**

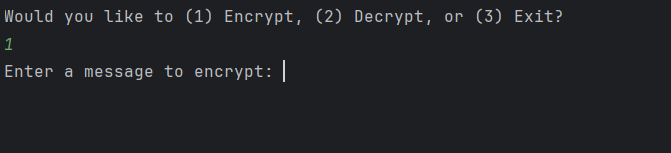
Run program

**A screen shot of a computer

Description automatically generated**

**Step 02**

Choice option to encrypt

****

**Step 03**

Enter massage to encrypt

**A black background with white text

Description automatically generated**

**Step 04**

Output encrypted massage and encrypted key

A black background with letters and numbers

Description automatically generated

**Decryption Process**

**Step 01**

Choice option to decrypt

**A black background with white text

Description automatically generated**

**Step 02**

Enter encrypted massage and encrypted key

A black screen with white text and numbers

Description automatically generated

**Step 03**

Output decrypted massage

A computer screen with text

Description automatically generated