Authentication Mechanism for IoT Infrastructure

B. Tech. Sem. – VII, Mini Project
Dept. of Computer Science & Engineering

By

Vinay Omkumar Khilwani	**18BCP126**
Shreedhar Manish Bhatt	**18BCP150D**

Under the Supervision
Of
Dr. Nishant Doshi

SCHOOL OF TECHNOLOGY
PANDIT DEENDAYAL ENERGY UNIVERSITY
GANDHINAGAR, GUJARAT, INDIA
July – December, 2021

Abstract

Recently, the use of Internet of Things (IoT) Infrastructure has tremendously increased. These difficult COVID-19 times have given a major push to decrease the risk of spreading diseases by limiting human interaction, in turn, relying majorly on IoT and cloud based technologies, in different fields, such as, transportation, healthcare, to name a few. As the reliability and demand for IoT Infrastructure is increasing, there is a definite need to secure the transfer of sensitive information via insecure public channels, with highest achievable efficiency. For the same, *Rana et. al.* proposed a light-weight user authentication scheme for IoT Infrastructure. We have attentively analysed the scheme of *Rana et. al.*, and found out that it is insecure against some attacks, including Server Compromise Attack, Stolen Smart Card (Device) Attack, User Impersonation Attack and Password Change Attack. In turn, we have proposed our own scheme, which is secure against the mentioned attacks. Our scheme is a good fit for high scale IoT systems, since it is light-weight and secure. Finally, we have compared our scheme with various existing schemes, in terms of security against several possible attacks, as well as computation and communication cost.

Table of Contents

1. Introduction

Internet of Things (IoT) and Cloud Based Infrastructure is all set to replace the current data and communication infrastructure. In recent times, IoT has been widely used to provide services to the general public, as well as various industries of the society [1] [2]. An IoT Infrastructure is a collection of numerous devices / sensors, set up at various places, to communicate with the server database (cloud server). IoT Infrastructure, put together with Cloud Based Infrastructure, is meant to communicate, store and process data.

The quantity of the IoT and cloud based systems is increasing day by day, especially in these COVID-19 times. The other reason for such popularity and use, is its comparatively lower deployment cost and varying and promising IoT devices and sensors. [3]

The real time, efficient, secure and accurate access of data in these systems is the key, which helps organisations to manage their resources effectively. In these difficult pandemic times of COVID-19, these systems are proving to be of definite use, since, they are also useful in reducing human involvement. [4] Hence, the use of such IoT systems is highly motivated.

Since, there is a growth in the demand of usage of such systems, there is a very crucial need to secure the data communication happening between the entities, using insecure public channels. Authentication between the communicating entities is a key requirement to secure the communication. The fruitfulness of these systems will come to reality, only if the security and privacy of users, organisations and institutions is not compromised.

1.1 Motivation

Kaul and Awasthi [5] presented a remote user authentication system based on smart cards, in 2016, in which a client can authenticate with a remote server. In 2021, the review of Kaul and Awasthi's scheme was done by *Rana et al.* [6], and in turn, *Rana et al.* proposed their new improved scheme, which is more secure than *Kaul and Awasthi*'s scheme. But, deeply analysing the scheme of *Rana et al.*, we found that their scheme is insecure against various serious attacks, namely, Server Compromise Attack, Stolen Smart Card (Device) Attack, User Impersonation Attack, and Password Change Attack. Hence, we were motivated to propose a new scheme, which is lightweight and more secure than *Rana et al.*'s scheme.

1.2 Contribution

- We have highlighted and pointed out that the scheme of *Rana et al.* [6] is susceptible as well as weak against several basic attacks.
- In turn, we have proposed our own scheme, which is lightweight, highly scalable, and more secure and resistant towards basic and advanced attacks, which will help to provide convenient and effective authorised access to users of the particular IoT system.
- We have implemented both schemes (*Rana et al.*'s scheme and our proposed scheme) in Python 3.9 to carefully analyse them.

2. Literature Survey

- *Lamport* [7] was the first one to propose a authenticated scheme on an insecure communication channel, which was based on verification tables and verification passwords, in 1981. Verification Tables are unable to provide the amount of security, which the recent and current IoT systems require.
- Keeping efficiency and security as the foremost priority, many authentication schemes [8-11] have been proposed and presented.
- *Das et al.* [12] proposed a pseudo ID based scheme, in 2004. It was a new idea, but it was insecure against several attacks.
- *Wang et al.* [13] presented a new improved scheme based on *Das et al.*'s scheme, which resisted their weaknesses and was useful on high scale.
- Further, *Chang et al.* [14] found and presented weaknesses in *Wang et al.*'s scheme, and in turn, presenting an improved scheme.
- *Kumari et al.* [15] found out that *Chang et al.*'s scheme is insecure against insider and server impersonation attacks. As a result, *Kumari et al.* [15], proposed an updated scheme, which is resistant to the aforementioned attacks.
- *Kaul and Awasthi* [5] presented that *Kumari et al.*'s scheme is still weak and the session key between the server and the user can be found out by an adversary ($\boldsymbol{A}$). Hence, they proposed an updated scheme to counter the vulnerabilities.
- *Rana et al.* [6] reviewed *Kaul and Awasthi*'s scheme to find out its weaknesses and introduced a new scheme for the same.
- In 2017, *Challa et al.* [16] introduced a new user authentication scheme, using digital signature. Their scheme provides untraceability feature. But, their scheme has high computation cost.

3. Review of *Rana et al.*'s [6] Scheme

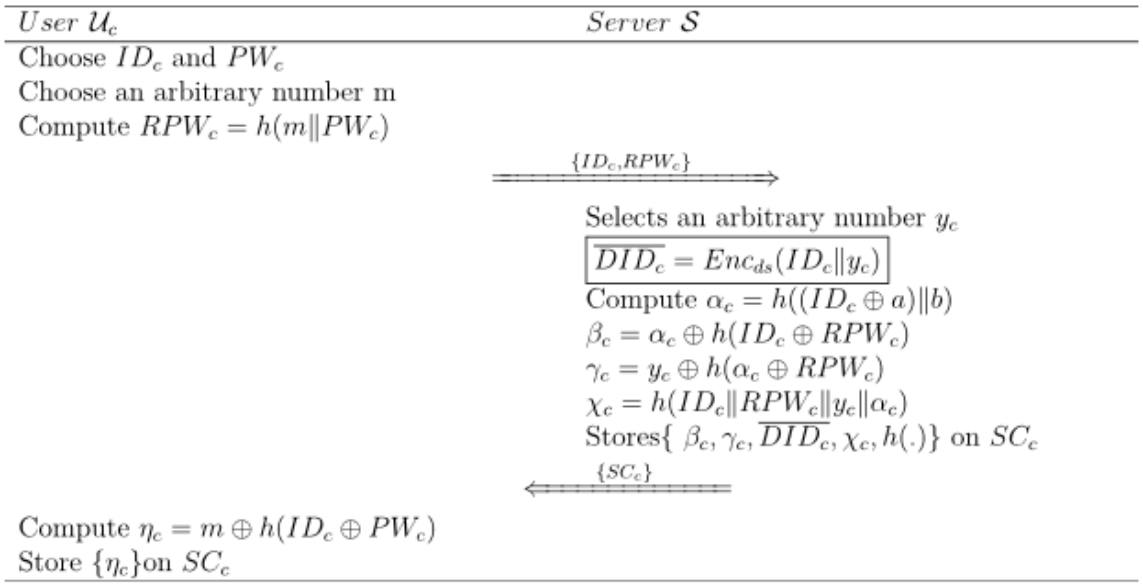
- We have demonstrated Rana et al.'s scheme in this section. We have presented the two phases of their scheme: Registration Phase & Login and Authentication Phase.
- Major terms of the scheme are described in Fig. 1

Symbols	Detail	Symbols	Detail
$\mathcal{U}_c$	cth legal user	ID_c	cth user identity
PW_c	cth user password	$\mathcal{S}$	Legal server
a, b	Private key and number of server	y_c	Arbitrary number for $\mathcal{U}_c$
SC_c	User's Smart Card	T_1	Time stamp obtained at User's side
T_2	Server's current time stamp	T'	Threshold value
δT_c	Time of transmission delay	$\|$	Concatenation operator
$\oplus$	XoR operator	$h(\cdot)$	Non-collision hash function
SK	Session key	$\mathcal{A}$	Adversary
$\Rightarrow$	Private communication channel	$\rightarrow$	Public communication channel

Fig. 1 Symbols

3.1 Registration Phase

- The registration phase of the scheme of Rana et al. is described in Fig. 2
- In this phase, the Server ***S*** registers a user ***U***
- The Server ***S*** doesn't store anything.



User $\mathcal{U}_c$ | Server $\mathcal{S}$

Choose ID_c and PW_c

Choose an arbitrary number m

Compute $RPW_c = h(m\|PW_c)$

$\xRightarrow{\{ID_c, RPW_c\}}$

Selects an arbitrary number y_c

$\boxed{\overline{DID_c} = Enc_{ds}(ID_c\|y_c)}$

Compute $\alpha_c = h((ID_c \oplus a)\|b)$

$\beta_c = \alpha_c \oplus h(ID_c \oplus RPW_c)$

$\gamma_c = y_c \oplus h(\alpha_c \oplus RPW_c)$

$\chi_c = h(ID_c\|RPW_c\|y_c\|\alpha_c)$

Stores$\{\beta_c, \gamma_c, \overline{DID_c}, \chi_c, h(.)\}$ on SC_c

$\xLeftarrow{\{SC_c\}}$

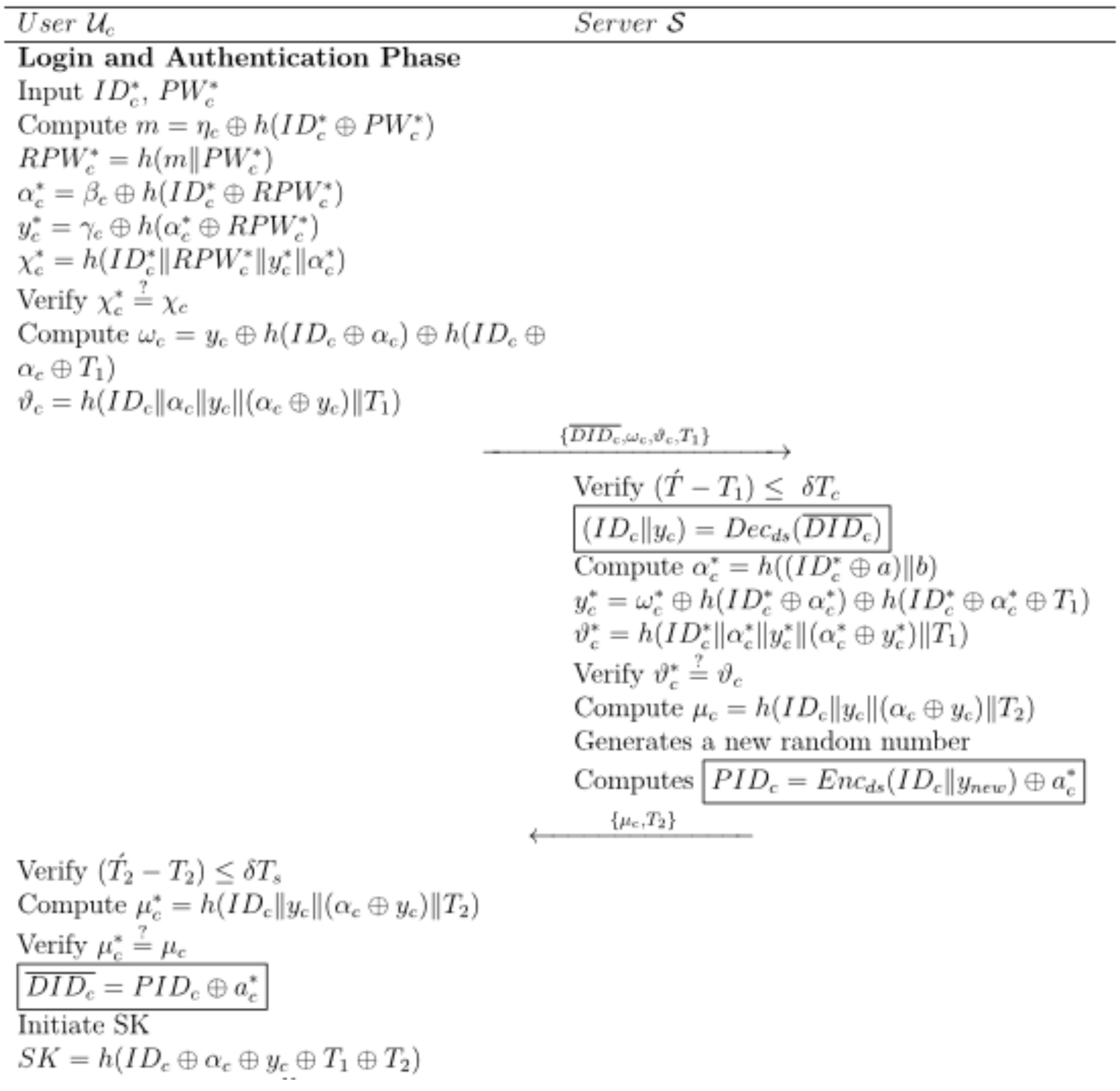
Compute $\eta_c = m \oplus h(ID_c \oplus PW_c)$

Store $\{\eta_c\}$on SC_c

Fig. 2 Registration Phase (Rana et al.)

3.2 Login and Authentication Phase

- The login and authentication phase of the scheme of Rana et al. is described in Fig. 3
- In this phase, firstly, the User ***U_c*** enters its ***ID_c*** and ***PW_c***
- Then, the server authenticates the User ***U_c***. If it is authenticated, the Session Key is generated by both communicating parties, i.e., Server and User, & the communication is established securely.



User $\mathcal{U}_c$ | *Server* $\mathcal{S}$

Login and Authentication Phase

Input ID_c^*, PW_c^*
Compute $m = \eta_c \oplus h(ID_c^* \oplus PW_c^*)$
$RPW_c^* = h(m \| PW_c^*)$
$\alpha_c^* = \beta_c \oplus h(ID_c^* \oplus RPW_c^*)$
$y_c^* = \gamma_c \oplus h(\alpha_c^* \oplus RPW_c^*)$
$\chi_c^* = h(ID_c^* \| RPW_c^* \| y_c^* \| \alpha_c^*)$
Verify $\chi_c^* \overset{?}{=} \chi_c$
Compute $\omega_c = y_c \oplus h(ID_c \oplus \alpha_c) \oplus h(ID_c \oplus \alpha_c \oplus T_1)$
$\vartheta_c = h(ID_c \| \alpha_c \| y_c \| (\alpha_c \oplus y_c) \| T_1)$

$\xrightarrow{\{\overline{DID_c}, \omega_c, \vartheta_c, T_1\}}$

Verify $(\acute{T} - T_1) \leq \delta T_c$
$\boxed{(ID_c \| y_c) = Dec_{ds}(\overline{DID_c})}$
Compute $\alpha_c^* = h((ID_c^* \oplus a) \| b)$
$y_c^* = \omega_c^* \oplus h(ID_c^* \oplus \alpha_c^*) \oplus h(ID_c^* \oplus \alpha_c^* \oplus T_1)$
$\vartheta_c^* = h(ID_c^* \| \alpha_c^* \| y_c^* \| (\alpha_c^* \oplus y_c^*) \| T_1)$
Verify $\vartheta_c^* \overset{?}{=} \vartheta_c$
Compute $\mu_c = h(ID_c \| y_c \| (\alpha_c \oplus y_c) \| T_2)$
Generates a new random number
Computes $\boxed{PID_c = Enc_{ds}(ID_c \| y_{new}) \oplus a_c^*}$

$\xleftarrow{\{\mu_c, T_2\}}$

Verify $(\acute{T_2} - T_2) \leq \delta T_s$
Compute $\mu_c^* = h(ID_c \| y_c \| (\alpha_c \oplus y_c) \| T_2)$
Verify $\mu_c^* \overset{?}{=} \mu_c$
$\boxed{\overline{DID_c} = PID_c \oplus a_c^*}$
Initiate SK
$SK = h(ID_c \oplus \alpha_c \oplus y_c \oplus T_1 \oplus T_2)$

Fig. 3 Login and Authentication Phase (Rana et al.)

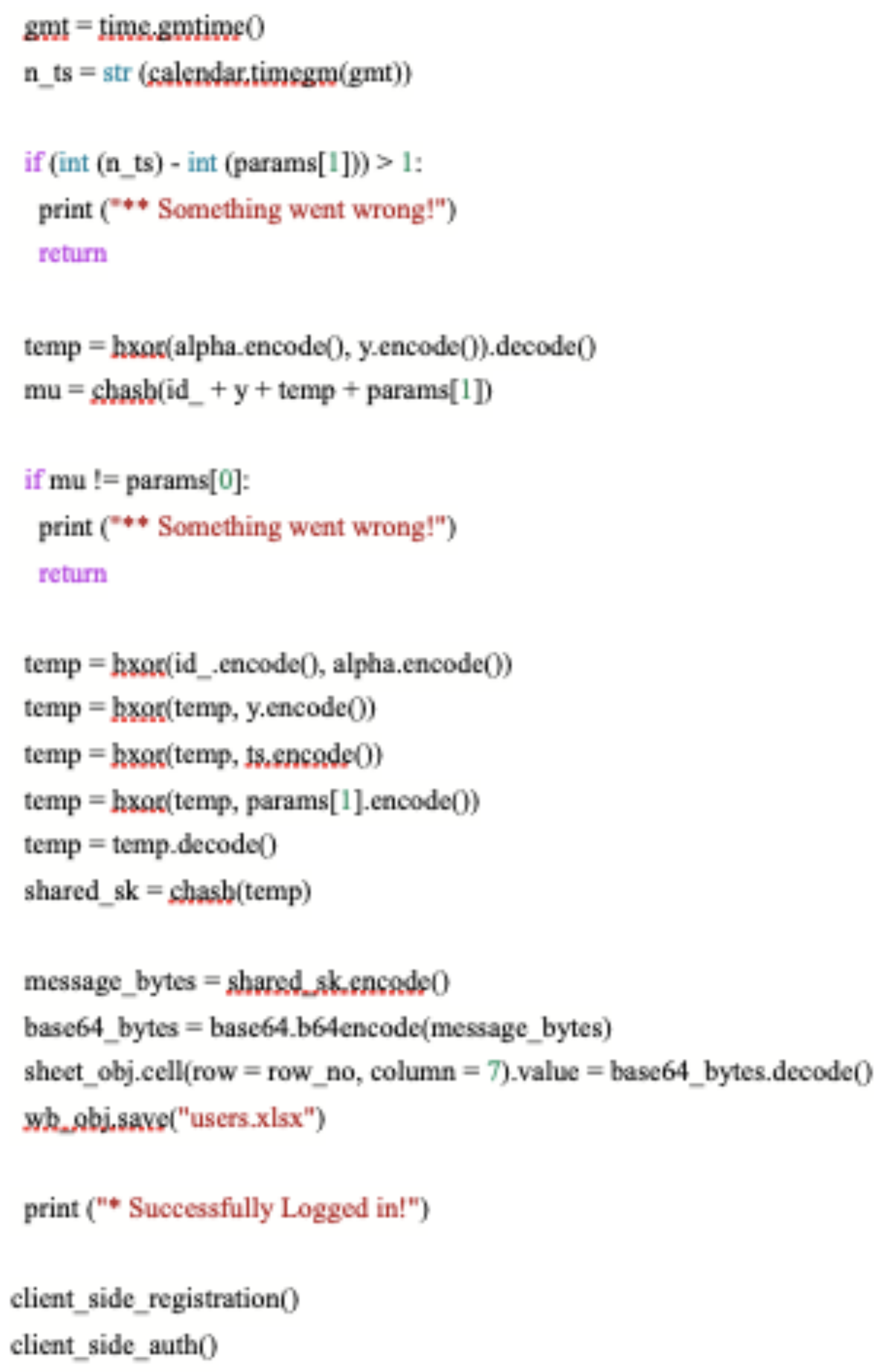
3.3 Implementation (Code) (Python 3.9)

```
import random
import calendar
import time
import openpyxl
import hashlib
from Crypto import Random
from Crypto.Cipher import AES
import base64
from base64 import b64encode, b64decode


def chash(s):
 hash = hashlib.sha256(s.encode()).hexdigest()
 return hash


class AESCipher(object):
 def __init__(self, key):
  self.block_size = AES.block_size
  self.key = hashlib.sha256(key.encode()).digest()

 def encrypt(self, plain_text):
  plain_text = self.__pad(plain_text)
  iv = Random.new().read(self.block_size)
  cipher = AES.new(self.key, AES.MODE_CBC, iv)
  encrypted_text = cipher.encrypt(plain_text.encode())
  return b64encode(iv + encrypted_text).decode("utf-8")

 def decrypt(self, encrypted_text):
  encrypted_text = b64decode(encrypted_text)
  iv = encrypted_text[:self.block_size]
  cipher = AES.new(self.key, AES.MODE_CBC, iv)
  plain_text = cipher.decrypt(encrypted_text[self.block_size:]).decode("utf-8")
  return self.__unpad(plain_text)

 def __pad(self, plain_text):
  number_of_bytes_to_pad = self.block_size - len(plain_text) % self.block_size
  ascii_string = chr(number_of_bytes_to_pad)
  padding_str = number_of_bytes_to_pad * ascii_string
  padded_plain_text = plain_text + padding_str
```

```
    return padded_plain_text

  @staticmethod
  def __unpad(plain_text):
    last_character = plain_text[len(plain_text) - 1:]
    return plain_text[:-ord(last_character)]


def bxor(ba1, ba2):
    return bytes([_a ^ _b for _a, _b in zip(ba1, ba2)])


def server_side_registration(id, rpw):

  # generate random secret y
  y = str (random.randrange(1000000, 9999999, 1))

  # get server secrets
  file = open("server_secrets.txt", "r")
  a = file.readline()
  b = file.readline()
  ks = file.readline()
  a = a[:-1]
  b = b[:-1]

  aes_obj = AESCipher(ks)

  d_id = aes_obj.encrypt(id + y)

  temp = bxor(id.encode(), a.encode())
  alpha = chash(temp.decode() + b)

  temp = bxor(id.encode(), rpw.encode())
  temp = chash(temp.decode())
  beta = bxor(alpha.encode(), temp.encode())
  beta = beta.decode()

  temp = bxor(alpha.encode(), rpw.encode())
```

```
 temp = chash(temp.decode())
 gamma = bxor(y.encode(), temp.encode())
 gamma = gamma.decode()

 psi = chash(id + rpw + y + alpha)

 return [beta, gamma, d_id, psi]


def client_side_registration():

 # input id password for registration
 print ("--> Enter ID: ", end = '')
 id_ = input()
 print ("--> Enter Password: ", end = '')
 password_ = input()
 print ("--> Confirm Password: ", end = '')
 confirm_password_ = input()

 # check condition
 if password_ != confirm_password_ :
  print ("** Password & Confirm Password don't match! Please try again!")
  return

 # generate random secret m
 m = str (random.randrange(2874284, 764517642548164562, 1))

 # calculate rpw
 rpw = chash(str (m) + password_)

 params = server_side_registration(id_, rpw)

 wb_obj = openpyxl.load_workbook("users.xlsx")
 sheet_obj = wb_obj.active
 rows = sheet_obj.max_row
 ok = True
 for i in range(1, rows + 1):
  if sheet_obj.cell(row = i, column = 6).value == id_:
      ok = False
      break

  if ok == True:
    temp = chash(id_ + password_)
    eta = bxor(m.encode(), temp.encode()).decode()
    params.append(eta)
    for i in range(1, rows + 1):
      if sheet_obj.cell(row = i, column = 3).value == None:
        ind = int (1)
        for x in params:
          message_bytes = x.encode()
          base64_bytes = base64.b64encode(message_bytes)
          sheet_obj.cell(row = i, column = ind).value = base64_bytes.decode()
          ind += 1
        sheet_obj.cell(row = i, column = ind).value = id_
        break
    wb_obj.save("users.xlsx")
    print ("** Succefully registered!")

  else:
    print ("** Username already exists!")


def server_side_auth(d_id, omega, theta, ts):
  gmt = time.gmtime()
  ts_server = str (calendar.timegm(gmt))
  if (int (ts_server) - int (ts)) > 1:
    print ("** Something went wrong!")
    return []

  # get server secrets
  file = open("server_secrets.txt", "r")
  a = file.readline()
  b = file.readline()
  ks = file.readline()
  a = a[:-1]
  b = b[:-1]
```

```
 aes_obj = AESCipher(ks)

 extracted = aes_obj.decrypt(d_id)

 id = extracted[:-7]
 y = ""
 for i in range (len(extracted) - 7, len(extracted)):
   y += extracted[i]

 temp = bxor(id.encode(), a.encode()).decode() + b
 alpha = chash(temp)
 temp = bxor(alpha.encode(), y.encode()).decode()
 theta_n = chash(id + alpha + y + temp + ts)

 if theta != theta_n:
   print ("** Something went wrong!")
   return []

 mu = chash(id + y + temp + ts_server)

 return [mu, ts_server]

def client_side_auth():
 print ("--> Enter ID: ", end = '')
 id_ = input()
 print ("--> Enter Password: ", end = '')
 password_ = input()

 wb_obj = openpyxl.load_workbook("users.xlsx")
 sheet_obj = wb_obj.active
 rows = sheet_obj.max_row
 ok = False
 row_no = -1
 for i in range(1, rows + 1):
   if sheet_obj.cell(row = i, column = 6).value == id_:
     row_no = i
     ok = True
   break

if ok == False:
  print ("** ID doesn't exist!")
  return

beta = base64.b64decode(sheet_obj.cell(row = row_no, column = 1).value.encode()).decode()
gamma = base64.b64decode(sheet_obj.cell(row = row_no, column = 2).value.encode()).decode()
d_id = base64.b64decode(sheet_obj.cell(row = row_no, column = 3).value.encode()).decode()
psi = base64.b64decode(sheet_obj.cell(row = row_no, column = 4).value.encode()).decode()
eta = base64.b64decode(sheet_obj.cell(row = row_no, column = 5).value.encode()).decode()

m = bxor(eta.encode(), chash(id_ + password_).encode()).decode()
rpw = chash(m + password_)
alpha = bxor(beta.encode(), chash(bxor(id_.encode(), rpw.encode()).decode()).encode()).decode()
y = bxor(gamma.encode(), chash(bxor(alpha.encode(), rpw.encode()).decode()).encode()).decode()
psi_n = chash(id_ + rpw + y + alpha)

if psi_n != psi:
  print ("** Something went wrong!")
  return

gmt = time.gmtime()
ts = str (calendar.timegm(gmt))

temp = bxor(id_.encode(), alpha.encode()).decode()
omega = bxor(y.encode(), chash(temp).encode()).decode()
temp = bxor(temp.encode(), ts.encode()).decode()
omega = bxor(omega.encode(), chash(temp).encode()).decode()

temp = bxor(alpha.encode(), y.encode()).decode()
theta = chash(id_ + alpha + y + temp + ts)

params = server_side_auth(d_id, omega, theta, ts)

if (len(params) == 0):
  return
```



```
    gmt = time.gmtime()
    n_ts = str (calendar.timegm(gmt))

    if (int (n_ts) - int (params[1])) > 1:
        print ("** Something went wrong!")
        return

    temp = bxor(alpha.encode(), y.encode()).decode()
    mu = chash(id_ + y + temp + params[1])

    if mu != params[0]:
        print ("** Something went wrong!")
        return

    temp = bxor(id_.encode(), alpha.encode())
    temp = bxor(temp, y.encode())
    temp = bxor(temp, ts.encode())
    temp = bxor(temp, params[1].encode())
    temp = temp.decode()
    shared_sk = chash(temp)

    message_bytes = shared_sk.encode()
    base64_bytes = base64.b64encode(message_bytes)
    sheet_obj.cell(row = row_no, column = 7).value = base64_bytes.decode()
    wb_obj.save("users.xlsx")

    print ("* Successfully Logged in!")

client_side_registration()
client_side_auth()
```

3.4 Cryptanalysis of the Scheme

1. Server Compromise Attack

- If the server's secret keys ***a*** and ***b*** are compromised, and messages ***M1 (***$\{DID_c, \omega_c, \vartheta_c, T_1\}$***)*** and ***M2 (***$\{\mu_c, T_2\}$***)*** are known to the adversary ***A***, then the session key can easily be found out by ***A***.

- Since DID_c is known to ***A***, he/she can find ID_c and y_c by the following equation:

$$(ID_c \,||\, y_c) = Dec_{ds}(DID_c)$$

- *T1* and *T2* are known to ***A***, since he/she has the access to messages ***M1*** ($\{DID_c, \omega_c, \vartheta_c, T_1\}$) and ***M2*** ($\{\mu_c, T_2\}$).
- Hence, the session key SK can be found out easily by ***A***, using the following:

$$SK = h(ID_c \oplus \alpha_c \oplus y_c \oplus T1 \oplus T2)$$

2. Stolen Smart Card (Device) Attack

- An adversary ***A*** can steal and gain unprivileged access to a user's smart card (device).
- After getting the unauthorised access, it can guess and find the password of the user, with the following steps:

  1. Suppose, during the registration of User *c*, the adversary ***A*** has got access to the insider information that was transferred via the private channel to the Server ***S*** $\{ID_C, RPW_C\}$
  2. Since, ***A*** has the stolen smart card (device) of the user *c*, he/she can extract all details of the card, via power analysis attacks. $\{\beta_c, \gamma_c, DID_c, \chi_c, \eta_c\}$
  3. Now, ***A*** can calculate the following:

     $\alpha_c^* = \beta_c \oplus h(ID_c \oplus RPW_c)$

     $y_c^* = \gamma_c \oplus h(\alpha_c^* \oplus RPW_c)$

     $\chi^*_c = h(ID_c \,||\, RPW_c \,||\, y_c^* \,||\, \alpha_c^*)$

  4. ***A*** will check whether $\chi^*_c = \chi_c$
  5. If yes, ***A*** will keep guessing a password, until he/she correctly finds it. To check the correctness of the guessed password PW_c^*, ***A*** will do the following:

     $m^* = \eta_c \oplus h(ID_c \oplus PW_c^*)$

     $RPW_c^* = h(m^* \,||\, PW_c^*)$

     If:

     $RPW_c^* = RPW_c$

     Then, the guessed password PW_c^* is the correct password of User *c*

3. User Impersonation Attack

- As described above, as an insider person, adversary ***A*** can have the following information with him/her:

  $\{ID_C, RPW_C\}$

  $\{\beta_c, \gamma_c, DID_c, \chi_c, \eta_c\}$

- ***A*** can act as a authorised user *c* and generate a new timestamp $\boldsymbol{TS_c^f}$ and can send message ***M1*** to the Server via the open channel.
- The Server ***S*** won't be able to differentiate the user *c* and adversary ***A***, and will continue its operations in the normal manner.
- In the end, user and server ***S*** will calculate the Shared Session Key (SK) using the required parameters.

- Finally, the adversary ***A*** will have the shared session key (SK) of the session, authenticated between the Server ***S*** and User *c*.

4. Password Change Attack

- The adversary ***A*** has the ability to find out the password of a User *c*, as described in the Stolen Smart Card (Device) Attack above.
- Hence, ***A*** can easily change the password of user *c*, using the <u>Password Change</u> mechanism of scheme of Rana et al., described in Fig. 4 below.

Input ID_c^*, PW_c^*, PW_c^N
Compute $m = \eta_c \oplus h(ID_c^* \oplus PW_c^*)$
$RPW_c^* = h(m \| PW_c^*)$
$\alpha_c^* = \beta_c \oplus h(ID_c^* \oplus RPW_c^*)$
$y_c^* = \gamma_c \oplus h(\alpha_c^* \oplus RPW_c^*)$
$\chi_c^* = h(ID_c^* \| RPW_c^* \| y_c^* \| \alpha_c^*)$
Verify $\chi_c^* \overset{?}{=} \chi_c$
Compute $RPW_c^N = h(m \| PW_c^N)$
$\beta_c^N = \alpha_c \oplus h(ID_c \oplus RPW_c)$
$\gamma_c^N = y_c \oplus h(\alpha_c \oplus RPW_c)$
$\chi_c^N = h(ID_c \| RPW_c \| y_c \| \alpha_c)$
$\eta_c^N = m \oplus h(ID_c \oplus PW_c)$
Update $\beta_c^N, \gamma_c^N, \chi_c^N, \eta_c^N$ on smart card

Fig. 4 Password Change Phase (Rana et al.'s Scheme)

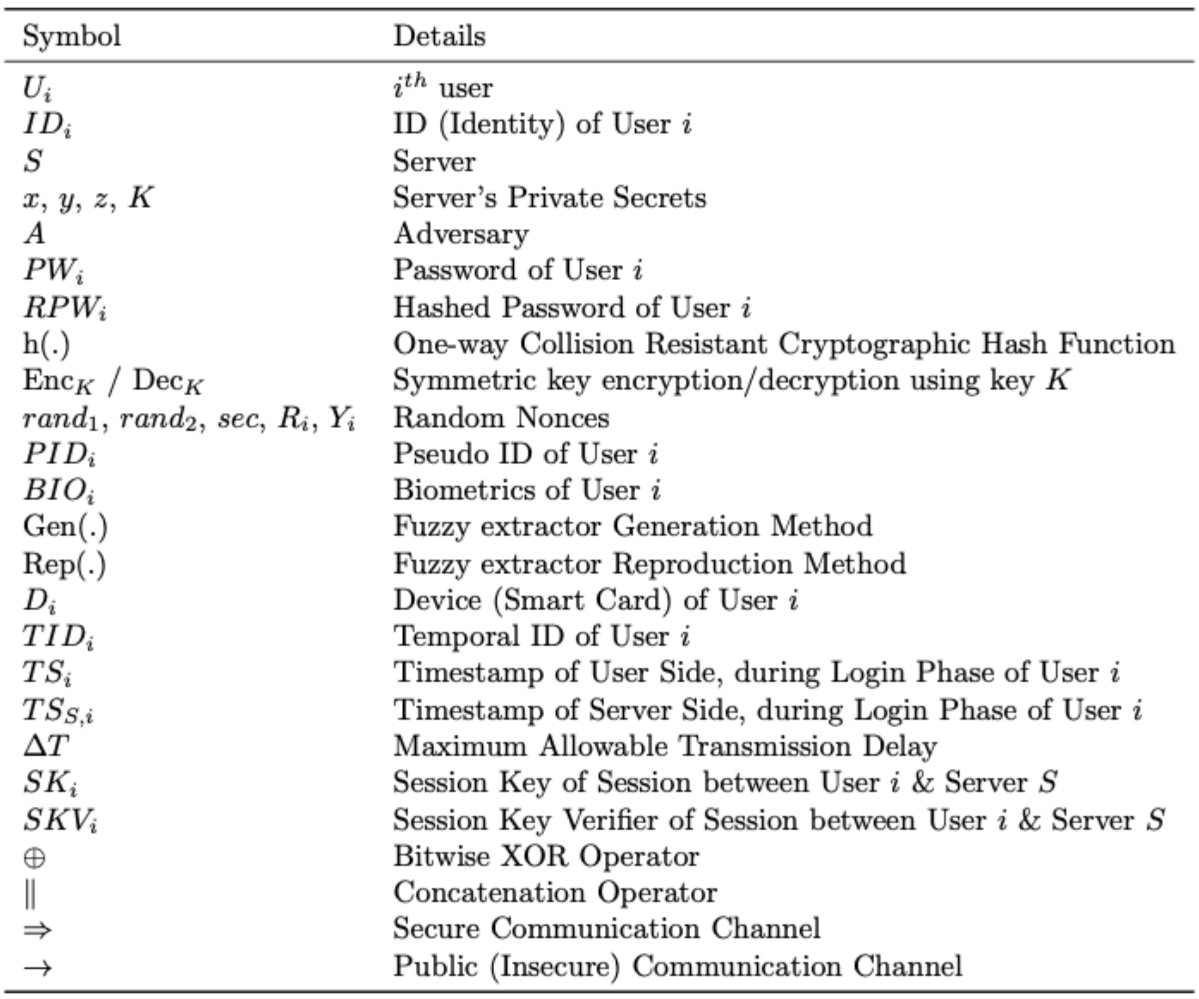
4. Our Proposed Scheme

- We have demonstrated our proposed and modified scheme in this section. We have presented the two phases of their scheme: Registration Phase & Login and Authentication Phase.
- We have used user's biometric information to ensure high level of security in our scheme. For the same, we have used a very famous biometrics scheme for its verification, which consists of following two functions:

$$Gen(BIO_{Usr_i}) = (\sigma_{Usr_i}, \tau_{Usr_i})$$

$$Rep(BIO'_{Usr_i}, \tau_{Usr_i}) = \sigma_{Usr_i}$$

- Major terms of the scheme are described in Fig. 5



Symbol	Details
U_i	i^{th} user
ID_i	ID (Identity) of User i
S	Server
x, y, z, K	Server's Private Secrets
A	Adversary
PW_i	Password of User i
RPW_i	Hashed Password of User i
h(.)	One-way Collision Resistant Cryptographic Hash Function
Enc_K / Dec_K	Symmetric key encryption/decryption using key K
$rand_1$, $rand_2$, sec, R_i, Y_i	Random Nonces
PID_i	Pseudo ID of User i
BIO_i	Biometrics of User i
Gen(.)	Fuzzy extractor Generation Method
Rep(.)	Fuzzy extractor Reproduction Method
D_i	Device (Smart Card) of User i
TID_i	Temporal ID of User i
TS_i	Timestamp of User Side, during Login Phase of User i
$TS_{S,i}$	Timestamp of Server Side, during Login Phase of User i
ΔT	Maximum Allowable Transmission Delay
SK_i	Session Key of Session between User i & Server S
SKV_i	Session Key Verifier of Session between User i & Server S
$\oplus$	Bitwise XOR Operator
$\|$	Concatenation Operator
$\Rightarrow$	Secure Communication Channel
$\rightarrow$	Public (Insecure) Communication Channel

Fig. 5 Symbols (Our Scheme)

4.1 Registration Phase

- The registration phase of our scheme is described in Fig. 6
- In this phase, the Server ***S*** registers a user ***U**$_i$* securely, and saves two parameters (TID, PID) of User *i* in its database.

Client (User i)	Server
Input ID_i & PW_i from User i Generate random nonces sec & R_i $RPW_i = \text{h}(PW_i \parallel sec)$ $RPW_i' = RPW_i \oplus R_i$ $M = \{TID_i, ID_i, RPW_i'\}$ $\Longrightarrow$	
	Generate a random nonce Y_i $PID_i = \text{Enc}_K\ (Y_i \parallel ID_i)$ $a_i = \text{h}(x \parallel (ID_i \oplus y) \parallel z)$ $b_i' = a_i \oplus RPW_i'$ $c_i' = Y_i \oplus RPW_i'$ $d_i' = \text{h}(a_i \parallel Y_i \parallel ID_i)$ Save Pair in Server's DB $\rightarrow \{TID_i,\ PID_i\}$ $D_i = \{TID_i,\ b_i',\ c_i',\ d_i'\}$ $\Longleftarrow$
$b_i = (b_i' \oplus RPW_i') \oplus \text{h}(\sigma_i \parallel ID_i \parallel RPW_i)$ $c_i = Y_i \oplus \text{h}(a_i \parallel \sigma_i \parallel ID_i \parallel RPW_i)$ $d_i = \text{h}(\sigma_i \parallel RPW_i \parallel d_i')$ $e_i = sec \oplus \text{h}(\sigma_i \parallel ID_i \parallel PW_i)$ Update $D_i = \{TID_i,\ b_i,\ c_i,\ d_i,\ e_i,\ \tau_i\}$	

Fig. 6 Registration Phase (Our Scheme)

4.2 Login and Authentication Phase

- The login and authentication phase of the scheme of Rana et al. is described in Fig. 7
- In this phase, firstly, the User ***U**$_i$* enters its ***ID**$_i$*, ***PW**$_i$*, ***BIO**$_i$*
- Then, the server authenticates the User ***U**$_i$*. If it is authenticated, the Session Key is generated by both communicating parties, i.e., Server and User, & the communication is established securely.

Client (User i)	Server
Input ID_i^*, PW_i^*, BIO_i^* Check whether ID_i^* exists? If NO, then STOP. $\sigma_i^* = \mathrm{Rep}(BIO_i^*, \tau_i)$ $sec = e_i \oplus h(\sigma_i^* \parallel ID_i^* \parallel PW^*)$ $RPW_i^* = h(PW_i^* \parallel \mathrm{sec})$ $a_i^* = b_i \oplus h(\sigma_i^* \parallel ID_i^* \parallel RPW_i^*)$ $Y_i^* = c_i \oplus h(a_i^* \parallel \sigma_i^* \parallel ID_i^* \parallel RPW_i^*)$ $d_i^* = \mathrm{h}(\sigma_i^* \parallel RPW_i^* \parallel \mathrm{h}(a_i^* \parallel Y_i^* \parallel ID_i^*))$ Check $d_i = d_i^*$? If NO, then STOP. Generate a Fresh User Timestamp TS_i Generate a random nonce $rand_1$ $rr_1 = \mathrm{h}(rand_1 \parallel TS_i) \oplus \mathrm{h}(Y_i^* \parallel a_i^* \parallel ID_i^* \parallel TS_i)$ $g_i = \mathrm{h}(a_i^* \parallel (a_i^* \oplus Y_i^*) \parallel Y_i^* \parallel rand_1 \parallel ID_i^* \parallel TS_i)$ $M_1 = \{TID_i, rr_1, g_i, TS_i\}$ $\longrightarrow$	
	Check whether TS_i^* - $TS_i \leq \Delta T$? If NO, then STOP. Fetch PID_i using TID_i from Server DB $(Y_i \parallel ID_i) = \mathrm{Dec}_K\ (PID_i)$ $a_i^* = \mathrm{h}(x \parallel (ID_i \oplus y) \parallel z)$ $rand_1^* = rr_1 \oplus \mathrm{h}(Y_i \parallel a_i^* \parallel ID_i \parallel TS_i)$ $g_i^* = \mathrm{h}(a_i^* \parallel (a_i^* \oplus Y_i) \parallel Y_i \parallel rand_1^* \parallel ID_i \parallel TS_i)$ Check $g_i = g_i^*$? If NO, then STOP. Generate a Fresh Server Timestamp $TS_{S,i}$ Generate a random nonce $rand_2$ Generate new temporal ID of User i, TID_i^{new} $rr_2 = \mathrm{h}(TS_{S,i} \parallel a_i^* \parallel Y_i \parallel ID_i) \oplus \mathrm{h}(rand_2 \parallel TS_{S,i})$ $t_i = \mathrm{h}(TS_{S,i} \parallel (a_i^* \oplus Y_i) \parallel rr_2 \parallel a_i^* \parallel Y_i \parallel ID_i)$ $SK_i = \mathrm{h}((TID_i \oplus TS_i \oplus TS_{S,i} \oplus ID_i \oplus a_i^* \oplus Y_i \oplus$ $\mathrm{h}(rand_2 \parallel TS_{S,i})) \parallel rand_1^*)$ $SKV_i = \mathrm{h}(TS_{S,i} \parallel SK_i \parallel rand_1^*)$ $TID_i^* = TID_i^{new} \oplus \mathrm{h}(TID_i \parallel TS_{S,i} \parallel SK_i)$ $M_2 = \{TID_i^*, t_i, rr_2, SKV_i, TS_{S,i}\}$ $\longleftarrow$
Check whether $TS_{S,i}^*$ - $TS_{S,i} \leq \Delta T$? If NO, then STOP. $t_i^* = \mathrm{h}(TS_{S,i} \parallel (a_i^* \oplus Y_i^*) \parallel rr_2 \parallel a_i^* \parallel Y_i^* \parallel ID_i^*)$ Check $t_i = t_i^*$? If NO, then STOP. $rand_2^* = rr_2 \oplus \mathrm{h}(TS_{S,i} \parallel a_i^* \parallel Y_i^* \parallel ID_i^*)$ $SK_i^* = \mathrm{h}((TID_i \oplus TS_i \oplus TS_{S,i} \oplus ID_i^* \oplus a_i^* \oplus Y_i^* \oplus rand_2^*) \parallel rand_1)$ $SKV_i^* = \mathrm{h}(TS_{S,i} \parallel SK_i^* \parallel rand_1)$ Check $SKV_i = SKV_i^*$? If NO, then STOP. $TID_i^{new} = TID_i^* \oplus \mathrm{h}(TID_i \parallel TS_{S,i} \parallel SK_i^*)$ Update TID_i with TID_i^{new} in D_i	
	Update TID_i with TID_i^{new} in Server's Database (S)

Fig. 7 Login & Authentication Phase (Our Scheme)

4.3 Implementation (Code) (Python 3.9)

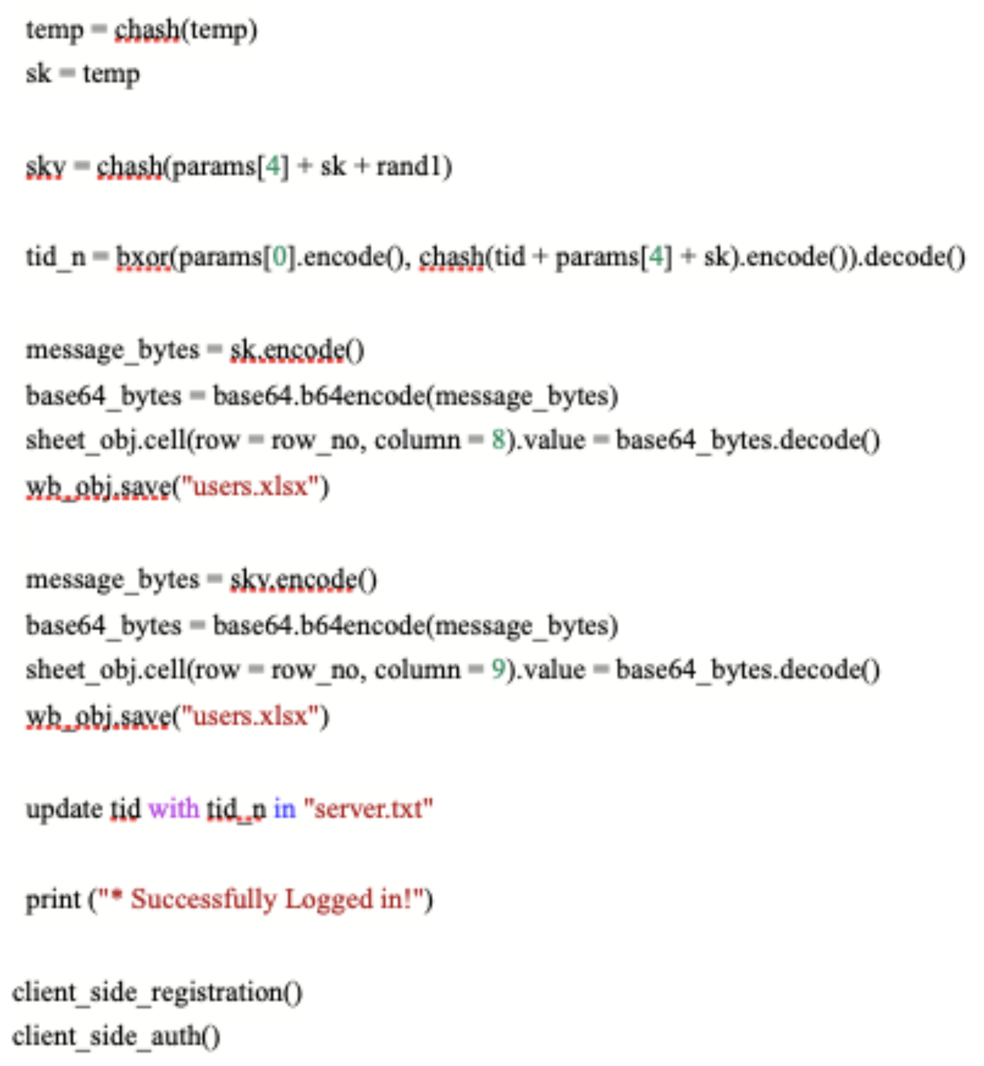
```
import random
import calendar
import time
import openpyxl
import hashlib
from Crypto import Random
from Crypto.Cipher import AES
import base64
from base64 import b64encode, b64decode


def chash(s):
  hash = hashlib.sha256(s.encode()).hexdigest()
  return hash


class AESCipher(object):
  def __init__(self, key):
    self.block_size = AES.block_size
    self.key = hashlib.sha256(key.encode()).digest()

  def encrypt(self, plain_text):
    plain_text = self.__pad(plain_text)
    iv = Random.new().read(self.block_size)
    cipher = AES.new(self.key, AES.MODE_CBC, iv)
    encrypted_text = cipher.encrypt(plain_text.encode())
    return b64encode(iv + encrypted_text).decode("utf-8")

  def decrypt(self, encrypted_text):
    encrypted_text = b64decode(encrypted_text)
    iv = encrypted_text[:self.block_size]
    cipher = AES.new(self.key, AES.MODE_CBC, iv)
    plain_text = cipher.decrypt(encrypted_text[self.block_size:]).decode("utf-8")
    return self.__unpad(plain_text)

  def __pad(self, plain_text):
    number_of_bytes_to_pad = self.block_size - len(plain_text) % self.block_size
    ascii_string = chr(number_of_bytes_to_pad)
    padding_str = number_of_bytes_to_pad * ascii_string
    padded_plain_text = plain_text + padding_str
    return padded_plain_text

  @staticmethod
  def __unpad(plain_text):
    last_character = plain_text[len(plain_text) - 1:]
    return plain_text[:-ord(last_character)]


def bxor(ba1, ba2):
    return bytes([_a ^ _b for _a, _b in zip(ba1, ba2)])


def server_side_registration(id, rpw, tid):

  # generate random secret y
  Y = str (random.randrange(1000000, 9999999, 1))

  # get server secrets
  file = open("server_secrets.txt", "r")
  a = file.readline()
  y = file.readline()
  z = file.readline()
  ks = file.readline()
  x = x[:-1]
  y = y[:-1]
  z = z[:-1]

  aes_obj = AESCipher(ks)

  p_id = aes_obj.encrypt(Y + id)

  a = chash(x + bxor(id.encode(), y.encode()).decode() + z)
  b_dash = bxor(a.encode(), rpw.encode()).decode()
  c_dash = bxor(Y.encode(), rpw.encode()).decode()
```

```
 d_dash = chash(a + Y + id)

 wb_obj = openpyxl.load_workbook("server.xlsx")
 sheet_obj = wb_obj.active
 rows = sheet_obj.max_row
 params = [tid, p_id]

 for i in range(1, rows + 1):
   if sheet_obj.cell(row = i, column = 1).value == None:
     ind = int (1)
     for x in params:
       message_bytes = x.encode()
       base64_bytes = base64.b64encode(message_bytes)
       sheet_obj.cell(row = i, column = ind).value = base64_bytes.decode()
       ind += 1
     break
 wb_obj.save("server.xlsx")

 return [b_dash, c_dash, d_dash]


def client_side_registration():

 # input id password for registration
 print ("--> Enter ID: ", end = '')
 id_ = input()
 print ("--> Enter Password: ", end = '')
 password_ = input()
 print ("--> Confirm Password: ", end = '')
 confirm_password_ = input()

 # check condition
 if password_ != confirm_password_ :
   print ("** Password & Confirm Password don't match! Please try again!")
   return

 # generate random secret m
 sec = str (random.randrange(2874284, 764517642548164562, 1))
```

```
# calculate rpw
rpw = chash(password_ + str (sec))
R = str (random.randrange(2874284, 764517642548164562, 1))
rpw_dash = bxor(rpw.encode(), R.encode()).decode()

tid = str(random.randrange(1, 20, 1))

params = server_side_registration(id_, rpw_hdash, tid)

params[0] = b_dash
params[1] = c_dash
params[2] = d_dash

sigma = str (random.randrange(2874284, 764517642548164562, 1))
tou = str (random.randrange(2874284, 764517642548164562, 1))
bio = sigma ^ tou

b = bxor(bxor(b_dash.encode(), rpw_dash.encode()), chash(sigma + id_ + rpw).encode()).decode()
c = bxor(c_dash.encode(), chash(a + sigma + id_ + rpw).encode()).decode()
d = chash(sigma + rpw + d_dash)
e = bxor(sec.encode(), chash(sigma + id_ + password_).encode()).decode()

wb_obj = openpyxl.load_workbook("users.xlsx")
sheet_obj = wb_obj.active
rows = sheet_obj.max_row
ok = True
for i in range(1, rows + 1):
  if sheet_obj.cell(row = i, column = 6).value == id_:
    ok = False
    break

if ok == True:
  params = [tid, b, c, d, e, tou]
  for i in range(1, rows + 1):
    if sheet_obj.cell(row = i, column = 3).value == None:
      ind = int (1)
      for x in params:
            message_bytes = x.encode()
            base64_bytes = base64.b64encode(message_bytes)
            sheet_obj.cell(row = i, column = ind).value = base64_bytes.decode()
            ind += 1
          sheet_obj.cell(row = i, column = ind).value = id_
          break
      wb_obj.save("users.xlsx")
      print ("** Succefully registered!")

    else:
      print ("** Username already exists!")


def server_side_auth(tid, rr1, g, ts):
    gmt = time.gmtime()
    ts_server = str (calendar.timegm(gmt))
    if (int (ts_server) - int (ts)) > 1:
      print ("** Something went wrong!")
      return []

    # get server secrets
    file = open("server_secrets.txt", "r")
    x = file.readline()
    y = file.readline()
    z = file.readline()
    ks = file.readline()
    x = x[:-1]
    y = y[:-1]
    z = z[:-1]

    pid = hash_map[tid]

    aes_obj = AESCipher(ks)

    extracted = aes_obj.decrypt(pid)

    id = extracted[7:]
    Y = ""
```

```
 for i in range (0, 7):
  Y += extracted[i]

 a = chash(x + bxor(id.encode(), y.encode()).decode() + z)
 rand1 = bxor(rr1.encode(), chash(Y + a + id + ts).encode()).decode()
 g_n = chash(a + bxor(a.encode(), Y.encode()).decode() + Y + rand1 + id + ts)

 if g != g_n:
  print ("** Something went wrong!")
  return []

 ts_server = str (calendar.timegm(gmt))
 rand2 = str (random.randrange(2874284, 764517642548164562, 1))
 tid_n = str(random.randrange(1, 20, 1))

 rr2 = bxor(chash(ts_server + a + Y + id).encode(), chash(rand2 + ts_server)).decode()

 t = chash(ts_server + bxor(a.encode(), Y.encode()).decode() + rr2 + a + Y + id)

 temp = bxor(tid.encode(), ts.encode()).decode()
 temp = bxor(temp.encode(), ts_server.encode()).decode()
 temp = bxor(temp.encode(), id.encode()).decode()
 temp = bxor(temp.encode(), a.encode()).decode()
 temp = bxor(temp.encode(), Y.encode()).decode()
 temp = bxor(temp.encode(), chash(rand2 + ts_server).encode()).decode()
 temp = temp + rand1
 temp = chash(temp)
 sk = temp

 skv = chash(ts_server + sk + rand1)

 tid_ = bxor(tid_n.encode(), chash(tid + ts_server + sk).encode()).decode()

 return [tid_, t, rr2, skv, ts_server]

def client_side_auth():
 print ("--> Enter ID: ", end = '')
 id_ = input()
```

```
print ("--> Enter Password: ", end = '')
password_ = input()

bio = input()

wb_obj = openpyxl.load_workbook("users.xlsx")
sheet_obj = wb_obj.active
rows = sheet_obj.max_row
ok = False
row_no = -1
for i in range(1, rows + 1):
  if sheet_obj.cell(row = i, column = 7).value == id_:
    row_no = i
    ok = True
    break

if ok == False:
  print ("** ID doesn't exist!")
  return

tid = base64.b64decode(sheet_obj.cell(row = row_no, column = 1).value.encode()).decode()
b = base64.b64decode(sheet_obj.cell(row = row_no, column = 2).value.encode()).decode()
c = base64.b64decode(sheet_obj.cell(row = row_no, column = 3).value.encode()).decode()
d = base64.b64decode(sheet_obj.cell(row = row_no, column = 4).value.encode()).decode()
e = base64.b64decode(sheet_obj.cell(row = row_no, column = 5).value.encode()).decode()
tou = base64.b64decode(sheet_obj.cell(row = row_no, column = 5).value.encode()).decode()

sigma = int (tou) ^ int (bio)
sigma = str (sigma)

sec = bxor(e.encode(), chash(sigma + id_ + password_).encode()).decode()
rpw = chash(password_ + sec)

a = bxor(b.encode(), chash(sigma + id + rpw).encode()).decode()
Y = bxor(c.encode(), chash(a + sigma + id_ + rpw).encode()).decode()
d_n = chash(sigma + rpw + chash(a + Y + id_))

if d_n != d:
  print ("** Something went wrong!")
  return

gmt = time.gmtime()
ts = str (calendar.timegm(gmt))
rand1 = str (random.randrange(2874284, 764517642548164562, 1))


rrl = bxor(chash(rand1 + ts).encode(), chash(Y + a + id_ + ts).encode()).decode()
g = chash(a + temp + Y + rand1 + id_ + ts)

params = server_side_auth(tid, rrl, g, ts)

if (len(params) == 0):
  return

gmt = time.gmtime()
n_ts = str (calendar.timegm(gmt))

if (int (n_ts) - int (params[4])) > 1:
  print ("** Something went wrong!")
  return

t = chash(params[4] + bxor(a.encode(), Y.encode()).decode() + params[2] + a + Y + id)

if t != params[1]:
  print ("** Something went wrong!")
  return

rand2 = bxor(params[2].encode(), chash().encode(params[4] + a + Y + id)).decode()

temp = bxor(tid.encode(), ts.encode()).decode()
temp = bxor(temp.encode(), params[4].encode()).decode()
temp = bxor(temp.encode(), id.encode()).decode()
temp = bxor(temp.encode(), a.encode()).decode()
temp = bxor(temp.encode(), Y.encode()).decode()
temp = bxor(temp.encode(), rand2.encode()).decode()
temp = temp + rand1
```



```
    temp = chash(temp)
    sk = temp

    sky = chash(params[4] + sk + rand1)

    tid_n = bxor(params[0].encode(), chash(tid + params[4] + sk).encode()).decode()

    message_bytes = sk.encode()
    base64_bytes = base64.b64encode(message_bytes)
    sheet_obj.cell(row = row_no, column = 8).value = base64_bytes.decode()
    wb_obj.save("users.xlsx")

    message_bytes = sky.encode()
    base64_bytes = base64.b64encode(message_bytes)
    sheet_obj.cell(row = row_no, column = 9).value = base64_bytes.decode()
    wb_obj.save("users.xlsx")

    update tid with tid_n in "server.txt"

    print ("* Successfully Logged in!")

client_side_registration()
client_side_auth()
```

4.4 Security Analysis of our Scheme

1. Server Compromise Attack

- Even if the server's secret keys ***x***, ***y*** and ***z*** are compromised, and messages ***M1*** and ***M2*** are known to the adversary ***A***, then too, the session key can't be found out by ***A***.
- The reason for the same is the inclusion of random nonces used by the User *i* & Server ***S*** to calculate the Shared Session Key (SK).
- Hence, our scheme is secure against Server Compromise Attack.

2. Stolen Smart Card (Device) Attack

- An adversary ***A*** can steal and gain unprivileged access to a user's smart card (device).

- After getting the unauthorised access, ***A*** can't guess the password or find out any other parameter, since for calculating any parameter, there is a requirement of biometrics of User *i* (***Bio**$_i$*) to be known.
- Hence, in any way, ***A*** has no use of the stolen smart card.
- Thus, our scheme is efficient and safe against stolen smart card (device) attacks.

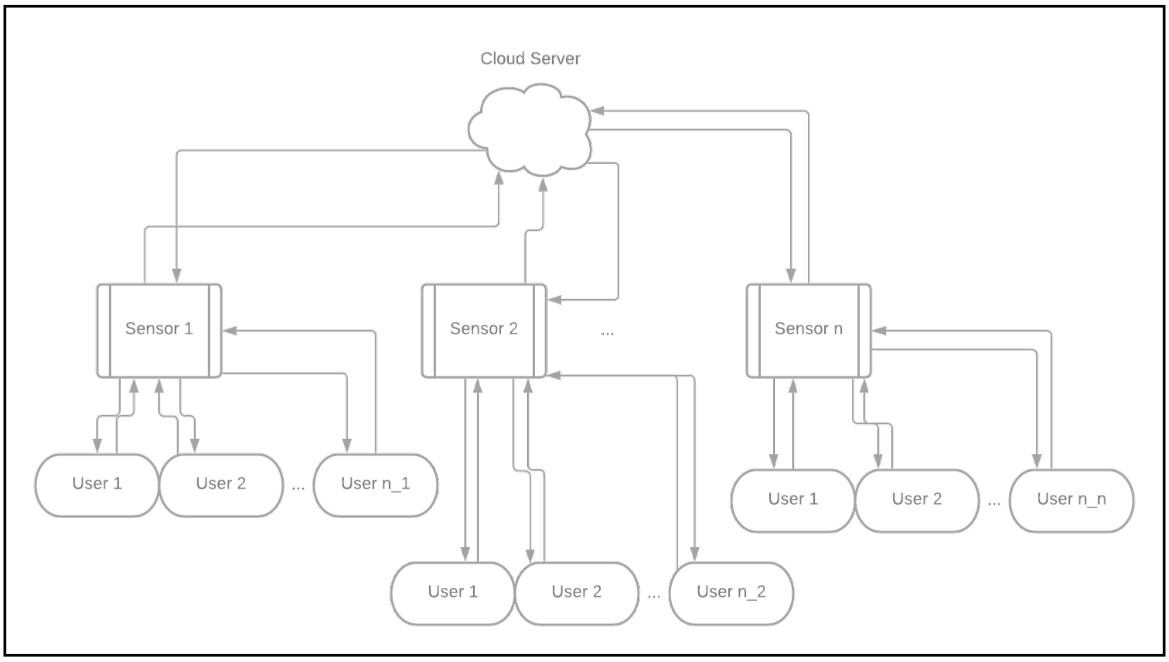
3. User Impersonation Attack

- ***A*** can act as a authorised user *i* and generate a new timestamp $\boldsymbol{TS_i^f}$
- But, the new timestamp generated by ***A*** will be of no use, as it can't gain access to any other parameter because of unavailability of biometrics of User *i* ***(Bio**$_i$**)***, hence, ***A*** won't be able to act as an authorised user to ***S***.
- Thus, our scheme is resistant to User Impersonation Attack.

4. Password Change Attack

- Since, ***A*** can't find the password of User *i* as described above, hence, ***A*** can't change the password of the user.
- Hence, our scheme is safe against Password Change Attack.

4.5 Network Diagram of General IoT Infrastructure, suitable for our scheme



5. Comparison with Existing Schemes

In this section, we will compare our scheme with some other existing schemes, with respect to Memory Cost, Communication Cost, Computation Cost and various possible attacks.

For implementation, we have used PyCrypto Library to implement hash functions in MacOS Catalina.

We have run our protocol several number of times to measure average computation cost. The average time taken for running collision-resistant hashing function is 0.00080 ms. The average time taken for concatenating is 0.00008 ms. The average times for other operations are negligible and hence aren't taken into account.

Memory Cost is the memory, in bits, required to store the parameters on the smart card (device) and server's database. In our scheme, we have stored 7 parameters in the smart card (device) and 2 parameters on the server. The memory cost of our scheme is slightly higher than the memory costs of Kaul and Awasthi, Kumari et al., Chang et al. and Rana et al.

T_h is defined as the time needed for computing hash function.
$T_{||}$ is defined as the time needed for computing concatenation.
$T_{\oplus}$ is define as the time required for computing XOR operation.

Communication Cost is the number of bits transferred via the channel during the login and authentication phase of the scheme.

Computation Cost is the time, in milliseconds (ms), required to run the login and authentication phase.

Scheme	Computation Cost	Memory Cost	Communication Cost
Ours	$24T_h + 55T_{\|\|} + 29T_{\oplus} = 0.0236$ ms	2304 bits	2560 bits
[6]	$20T_h + 27T_{\|\|} + 29T_{\oplus} = 0.01816$ ms	1536 bits	3296 bits
[5]	$20T_h + 27T_{\|\|} + 29T_{\oplus} = 0.01784$ ms	1280 bits	2668 bits
[14]	$20T_h + 27T_{\|\|} + 29T_{\oplus} = 0.01104$ ms	672 bits	2336 bits

Comparison Table

The Comparison Table based on various attacks is as follows:

Resistant to Attack	**Ours**	**[6]**	**[5]**	**[14]**
Server Compromise Attack	Yes	No	No	No
Password Guessing Attack	Yes	No	No	No
Stolen Smart Card Attack	Yes	No	Yes	Yes
Replay Attack	Yes	Yes	Yes	No
User Impersonation Attack	Yes	No	No	No
Password Change Attack	Yes	No	No	No
Man in the Middle Attack	Yes	Yes	Yes	Yes
Forward and Backward Secrecy	Yes	No	No	No

Looking at both the comparison tables, we can see that our scheme is little bit heavy on Memory Cost and Computation Cost as compared to other schemes, but, on the contrary, it is more safe and secure towards various attacks, compared to other related schemes.

6. Summary and Future Prospects

- To conclude, we have cryptically analysed the scheme of Rana et al. for lightweight authentication in IoT Infrastructure.
- We have proposed our new scheme, which is an improved version of Rana et al.'s scheme.
- We have proposed to use user biometrics to resist some possible easy attacks, in our scheme.
- We have tried to secure our scheme against the attacks mentioned in cryptanalysis of scheme of Rana et al.
- Other than that, we have finally compared our scheme to several other authentication schemes, in terms of various parameters and attacks.
- Our scheme have slight higher Memory Cost and Computational Cost than other schemes, but, on the other hand, our scheme is more secure towards basic and advanced cryptic attacks.
- Our scheme is scalable and lightweight and is easy to understand and implement.

- In the future, we want to work further on this scheme and improve it, if we find any other attacks that are possible on the same.
- Other than that, we want to publish this work of mini project as a research paper in a recent upcoming conference. Our preferable conference is MMCITRE-2022 (http://www.mmcitare.com/home.html). Its submission deadline is 01, December, 2021.

7. References

[1] Winkler, M., Tuchs, K.-D., Hughes, K., Barclay, G., 2008. Theoretical and practical aspects of military wireless sensor networks. J. Telecommun. Inf. Technol. 37–45.
[2] Kolokotsa, D., 2016. The role of smart grids in the building sector. Energy Build. 116, 703–708.
[3] N. Chen and M. Okada, "Towards 6G Internet of Things and the con- vergence with RoF system," *IEEE Internet Things J.*, early access, Dec. 28, 2020, doi: 10.1109/JIOT.2020.3047613.
[4] Lai, C.-C., Shih, T.-P., Ko, W.-C., Tang, H.-J., Hsueh, P.-R., 2020. Severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) and corona virus disease-2019 (COVID-19): The epidemic and the challenges. Int. J. Antimicrob. Ag. 105924.
[5] S.D. Kaul, A.K. Awasthi, Security enhancement of an improved remote user authentication scheme with key agreement, Wirel. Pers. Commun. 89 (2) (2016) 621–637.
[6] M. Rana, A. Shafiq, I. Altaf, M. Alazab, K. Mahmood, S. A. Chaudhry, and Y. B. Zikria, "A secure and lightweight authentication scheme for next generation IoT infrastructure," Comput. Commun., vol. 165, pp. 85–96, Jan. 2021.
[7] Lamport, L., 1981. Password authentication with insecure communication. Commun. ACM 24 (11), 770–772.
[8] Z. Zhang, Q. Qi, N. Kumar, N. Chilamkurti, H.-Y. Jeong, A secure authentication
scheme with anonymity for session initiation protocol using elliptic curve
cryptography, Multimedia Tools Appl. 74 (10) (2015) 3477–3488.
[9] Z. Ali, A. Ghani, I. Khan, S.A. Chaudhry, S.H. Islam, D. Giri, A robust authentication and access control protocol for securing wireless healthcare sensor
networks, J. Inf. Secur. Appl. 52 (2020) 102502.
[10] K. Mahmood, X. Li, S.A. Chaudhry, H. Naqvi, S. Kumari, A.K. Sangaiah, J.J. Rodrigues, Pairing based anonymous and secure key agreement protocol for smart grid edge computing infrastructure, Future Gener. Comput. Syst. 88 (2018) 491–500.
[11] S.A. Chaudhry, K. Yahya, F. Al-Turjman, M.-H. Yang, A secure and reliable device access control scheme for IoT based sensor cloud systems, IEEE Access 8 (2020) 139244–139254.
[12] M.L. Das, A. Saxena, V.P. Gulati, A dynamic ID-based remote user authentication scheme, IEEE Trans. Consum. Electron. 50 (2) (2004) 629–631.
[13] Y.-y. Wang, J.-y. Liu, F.-x. Xiao, J. Dan, A more efficient and secure dynamic ID-based remote user authentication scheme, Comput. Commun. 32 (4) (2009) 583–585.
[14] Y.-F. Chang, W.-L. Tai, H.-C. Chang, Untraceable dynamic-identity-based remote user authentication scheme with verifiable password update, Int. J. Commun. Syst. 27 (11) (2014) 3430–3440.
[15] S. Kumari, M.K. Khan, X. Li, An improved remote user authentication scheme with key agreement, Comput. Electr. Eng. 40 (6) (2014) 1997–2012.
[16] Challa, S., Wazid, M., Das, A.K., Kumar, N., Reddy, A.G., Yoon, E.-J., Yoo, K.-Y., 2017. Secure signature-based authenticated key establishment scheme for future IoT applications. IEEE Access 5, 3028–3043.