Report Titled

A Provably Secure Anonymous Biometrics-Based Authentication Scheme for Wireless Sensor Networks Using Chaotic Map

By

Name	ID No.
Subhash Reddy	2017A7PS0228H
Meher Gajula	2017AAPS0339H

Under Supervision of
Dr.Odelu Vanga
Department of Computer Science and Information Systems

Course - BITS F463 (Cryptography)
2019

Introduction

Wireless sensor network (WSN) refers to a group of spatially dispersed and dedicated sensors for monitoring and recording the physical conditions of the environment and organizing the collected data at a central location. WSNs measure environmental conditions like temperature, sound, pollution levels, humidity, wind, and so on. They typically have numerous sensor nodes and one or more gateway nodes, along with a large number of external users. The sensor nodes possess limited computation capability and storage space. They are deployed in unattended environments to collect valuable information and transmit the information to the gateway. The external users are able to access the gathered data with the help of the gateway.

The data is transmitted via the unprotected wireless channel. Hence, an effective protective measure should be adopted to protect the data from unauthorized access, illegal eavesdropping and tampering. So, we will be needing an authentication protocol which is a security mechanism that provides identity authentication and establishes a session key for the communication parties to realize secure data exchange.

However, the design of a practical authentication protocol for resource-constrained WSNs is not an easy task. The hash function based schemes have high efficiency, but it is hard to guarantee the security of the session key. A number of schemes adopting the public key cryptography such as ElGamal cryptography and Elliptic curve cryptography are introduced. However, these schemes involve a high computation cost. In addition to that, the hash function based schemes and the public key cryptosystem based schemes included, have design deficiencies and suffer

from various security attacks, like forgery attack, replay attack, and fail to provide user anonymity, etc.
There have been some Chebyshev chaotic map-based authentication protocols introduced. The intractability of the Chebyshev chaotic Diffie–Hellman problem (CHDHP) and its semigroup property make it feasible to establish a secure session key using Chebyshev chaotic map. Furthermore, the computation overhead of a Chebyshev polynomial is approximately 1/3 of scalar multiplication on the elliptic curve group. It significantly reduces the computing overhead and energy consumption for the resource-constrained sensor node.

Related works:

In 2009, the smart card authentication schemes for WSNs was used for the first time, but it has many security vulnerabilities such as sensor node impersonation attack and smart card lose attack. So, In 2014 Turkanovic introduced a two-factor authentication scheme for heterogeneous WSNs using a hash function. In 2015, Chang proposed a smart card-based dynamic identity authentication scheme employing the ElGamal cryptosystem but later it was proved that it is susceptible to user impersonation attack and offline guessing attack. In 2018, Amin presented a biometric-based scheme using hash function to enhance security, and in the same year, Aghili revealed that it is susceptible to desynchronization attack and proposed an improved scheme to remedy its vulnerability. In addition to his work, in this paper, we learn about an enhanced biometrics-based authentication scheme for WSNs using Chebyshev chaotic map. We discuss the security of the proposed scheme using several widely-accepted security analysis methods.

Implementation:

Programming Language used:

Python 3

Libraries used:

Random:

Since the algorithms needed to create random numbers for execution and input of Biometric is taken as a random number due to lack of Biometric I/O.

FuzzyExtractor:

Since the Biometric needed to be split into key and helper for the execution.

Hashlib:

To execute the hash functions in the given algorithms.

Math:

To convert the hexadecimal strings created during the hashing to int format for executing the XOR functions.

Time:

For checking the timestamps through each process. And for calculating the computing cost.

NaCl:

To execute the hash functions in the algorithms (using sha256).

Sys:

For termination of execution of the program when the credentials are wrong or when the time limit exceeds.

Binascii:

For conversion of hexadecimal strings to int values based on their ASCII values.

For multiple smart cards, a single smart card file was created and it stores the relevant variables in lists as in which works as multiple smart cards.

Conclusion:

The proposed scheme in the paper introduces a three-factor authentication in its algorithm, thus making it secure towards forgery attack, session key disclosure attack, forward secrecy attack, and sensor node impersonation attack.
Though the computation time cost of this proposed scheme is twice as that of Aghili's scheme, the space complexity is lesser and the security is much more enhanced.

CODE(Aghili's Scheme):

main.py:

```
from gwn import *
from SmartCard import *
import sys

gwn_obj = GWN()
smart_card_obj = SmartCard()
a = 0
b = 0
while True:

    user_input = int(input("BYE!!! -- 0\nREGISTRATION -- 1\nAUTH -- 2\n"))

    if user_input == 0:
        sys.exit()

    if user_input == 1:
        gwn_obj.ID.append(int(input("Enter your ID(8-digit number only)")))
        smart_card_obj.PW.append(int(input("Enter your password(8- digit number only)")))
        print("Imprint you biometric")
        smart_card_obj.B.append(random.randrange(999,9999,1))


    if user_input == 2:
        smart_card_obj.NewID = (int(input("Enter your ID(8-digit number only)")))
        smart_card_obj.NewPW = (int(input("Enter your password(8- digit number only)")))
        for i in gwn_obj.ID, smart_card_obj.PW:
            if (gwn_obj.ID[i] == smart_card_obj.NewID) and (smart_card_obj.PW[i] ==
smart_card_obj.NewPW):
                a = i
                break
            else :
                print("Wrong Credentials")
                sys.exit()
        print("Imprint you biometric")
        NewB = smart_card_obj.B[a]
        # print()
        b = int(input("Enter the node you want ot connect to (0 to 9)"))
        smart_card_obj
```

SensorNode.py:

```
import time
import hashlib
from main import a,b
from gwn import *

class SensorNode():

    def __init__(self):
        self.hid1 = []
        self.V = []
        self.D1 = []
        self.T3 = 0
        self.k4 = 0
        self.sk = 0
        self.T3 = 0
        self.k5 = 0
        self.W = []

    def main():
        a = main.a
        b = main.b
        T = time.time()
        if (T-T2 > 20):
            print("session timeout")
        s = str(e[b])+str(GWN.T2)
        hid1[a] = GWN.Y[a] ^ int(hashlib.sha256(s.encode('utf-8').hexdigest(),16))
        k3 = hid1[a] ^ V[a]
        s = str(hid1[a])+str(ej)+str(T2)+str(k3)
        D1[a] = hashlib.sha256(s.encode('utf-8').hexdigest())
        if D1[a] != GWN.D[a]:
            print("Authentication failed")
        T3 = time.time()
        k4 = random.randrange(999,9999,1)
        s = str(hid1[a])+str(sid[b])+str(k3)+str(k4)
        sk = hashlib.sha256(s.encode('utf-8').hexdigest())
        print(sk)
        s = str(sk) + str(T3)
        W[a] = hashlib.sha256(s.encode('utf-8').hexdigest())
        k5 = k3 ^ k4
```

gwn.py:

```
import time
import random
import hashlib
from main import a,b
from SmartCard import *
from SensorNode import *

class GWN_():

    def __init__(self):
        self.Masterkey = "1234567890"
        self.XG = int(self.Masterkey)
        self.sid = []
        self.e = []
        self.ID = []
        self.r1 = 0
        self.did = []
        self.f = []
        self.T1 = 0
        self.f1 = []
        self.k2 = 0
        self.hid = []
        self.sid1 = []
        self.N1 = []
        self.T2 = []
        self.didnew = []
        self.fnew = []
        self.D = []
        self.Y = []
        self.V = []
        self.N1 = []
        self.M0 = 0
        self.M2 = 0
        self.M3 = 0

    def main():
        for i in range (0,9):
            a = 1000 + i
            sid.append(a)
        for i in range (0,9):
            s = str(sid[a])+str(XG)
            e.append(hashlib.sha256(s.encode('utf-8')))
        r1 = random.randrange(999,9999,1)
```

```
        s = str(ID[a])+str(r1)
        SmartCard.did.append(hashlib.sha256(s.encode('utf-8')).hexdigest())
        s = str(did[a])+str(XG)
        SmartCard.f.append(hashlib.sha256(s.encode('utf-8')).hexdigest())
        a = main.a
        b = main.b

        T = time.time()
        if (T-T1 > 20):
            print("session timeout")
        s = str(did[a])+str(XG)
        f1[a] = hashlib.sha256(s.encode('utf-8')).hexdigest()
        s = str(did[a])+str(f[a])+str(ST1)
        k2 = L[a] ^ int(hashlib.sha256(s.encode('utf-8')).hexdigest(),16)
        s1 = str(ID[a])
        s = str(k2)+str(T1)
        hid[a] = Q[a] ^ int(hashlib.sha256(s.encode('utf-8')).hexdigest(),16)
        s = str(f1[a])+str(T1)
        sid1[b] = P[a] ^ int(hashlib.sha256(s.encode('utf-8')).hexdigest(),16)
        s = str(did[a])+str(k2)+str(f1[a])+str(T1)+str(sid1[b])+str(Q[a])
        N1[a] = hashlib.sha256(s.encode('utf-8')).hexdigest()

        if N1[a] != N[a]:
            print("Authentication failed")
        T2 = time.time()
        s = str(sid1[a])+str(initialisation.XG)
        e[b] = hashlib.sha256(s.encode('utf-8').hexdigest())
        s = str(int(hashlib.sha256(s1.encode('utf-8')).hexdigest(),16))+str(e[b])+str(T2)+str(k2)
        D[a] = hashlib.sha256(s.encode('utf-8').hexdigest())
        s = str(e[b])+str(T2)
        Y[a] = int(hashlib.sha256(s1.encode('utf-8').hexdigest(),16)) ^
int(hashlib.sha256(s.encode('utf-8').hexdigest(),16))
        V[a] = k2 ^ int(hashlib.sha256(s1.encode('utf-8').hexdigest(),16))

        T = time.time()
        if (T-T3 > 20):
            print("session timeout")
        k6 = k5 ^ k2
        s = str(hashlib.sha256(s1.encode('utf-8')))+str(sid[b])+str(k2)+str(k6)
        sk = hashlib.sha256(s.encode('utf-8').hexdigest())
        s = str(sk)+str(T3)
        W1[a] = hashlib.sha256(s.encode('utf-8').hexdigest())
        if W1[a] != W[a]:
```

```
            print("Authentication failed")
        T4 = time.time()
        s = str(sk)+str(k6)+str(T4)
        M0 = int(hashlib.sha256(s.encode('utf-8')).hexdigest(),16)
        s = str(sk)+str(k6)
        ID1[a] = SmartCard.M2 ^ int(hashlib.sha256(s.encode('utf-8')).hexdigest(),16)
        if hid1[a] != hid[a]:
            print("Error")
        r2 = random.randrange(999,9999,1)
        s = str(ID1[a])+str(r2)
        didnew[a] = hashlib.sha256(s.encode('utf-8')).hexdigest()
        s = str(didnew[a])+str(XG)
        fnew[a] = hashlib.sha256(s.encode('utf-8')).hexdigest()
        s = str(did[a])+str(k6)
        M3 = int(didnew[a],16) ^ int(hashlib.sha256(s.encode('utf-8')).hexdigest(),16)
        s = str(f[a])+str(k2)
        M4 = int(fnew[a],16) ^ int(hashlib.sha256(s.encode('utf-8')).hexdigest(),16)
        s1 = str(ID1[a])
        s = str(hashlib.sha256(s1.encode('utf-8')).hexdigest())+str(M3)+str(M4)
        M5 = hashlib.sha256(s.encode('utf-8')).hexdigest()


SmartCard.py:
import time
import random
import hashlib
from main import a,b
from fuzzy_extractor import FuzzyExtractor

class SmartCard():

    def __init__(self):
        self.PW = []
        self.B = []
        self.NewID = 0
        self.NewPW = 0
        self.NewB = 0
        self.did = []
        self.f = []
        self.C = []
        self.E = []
        self.A = []
        self.eta = []
```

```
        self.phi = []
        self.omega = []
        self.theta = []

    def main():
        extractor = FuzzyExtractor(16, 8)
        omegatemp, thetatemp = extractor.generate(B[i]) # key, helper
        omega.append(omegatemp)
        theta.append(thetatemp)
        s = str(GWN.ID[i])+str(PW[i])+str(B[i])
        A.append(hashlib.sha256(s.encode('utf-8')).hexdigest())
        s = str(GWN.ID[i])+str(PW[i])
        E.append(hex(int.from_bytes(theta[i], byteorder =
'little')^int(hashlib.sha256(s.encode('utf-8')).hexdigest(),16)))
        s = str(PW[i])+str(omega[i])
        C.append(int(f, 16)^int(hashlib.sha256(s.encode('utf-8')).hexdigest(),16))
        s = str(GWN.ID[i])+str(omega[i])
        eta.append(PW[i]^int(hashlib.sha256(s.encode('utf-8')).hexdigest(),16))
        phi.append(hashlib.sha256(s.encode('utf-8')).hexdigest())
        a = authentication.a
        b = authentication.b
        s = str(NewID)+str(NewPW)
        theta[a] = E[a] ^ int(hashlib.sha256(s.encode('utf-8')).hexdigest(),16)
        Newomega = extractor.reproduce(NewB,theta[a])
        s = str(NewPW)+str(Newomega)
        f[a] = C[a]^int(hashlib.sha256(s.encode('utf-8')).hexdigest(),16)
        s = str(NewID)+str(NewPW)+str(Newomega)
        A1[a] = hashlib.sha256(s.encode('utf-8')).hexdigest()
        if A[a] != A[a]:
        print("Authentication failed")

        k1 = random.randrange(999,9999,1)
        T1 = time.time()
        s = str(did[a])+str(f[a])+str(T1)
        L[a] = k1 ^ int(hashlib.sha256(s.encode('utf-8')).hexdigest(),16)
        s = str(f[a])+str(T1)
        P[a] = sid[a] ^ int(hashlib.sha256(s.encode('utf-8')).hexdigest(),16)
        s = str(k1)+str(T1)
        s1 = str(NewID)
        Q[a] = int(hashlib.sha256(s1.encode('utf-8')).hexdigest(),16) ^
int(hashlib.sha256(s.encode('utf-8')).hexdigest(),16)
        s = str(did[a])+str(k1)+str(f[a])+str(T1)+str(sid[a])+str(Q[a])
        N[a] = hashlib.sha256(s.encode('utf-8')).hexdigest()
```

```
T = time.time()
if (T-T4 > 20):
   print("session timeout")

k7 = k5 ^ k1
s = str(hashlib.sha256(s1.encode('utf-8')).hexdigest())+str(sid[b])+str(k1)+str(k7)
sk = hashlib.sha256(s.encode('utf-8')).hexdigest()
s = str(sk)+str(k7)+str(T4)
M1 = hashlib.sha256(s.encode('utf-8')).hexdigest()
if M1!=M0:
   print("Authentication failed")
elif M1 == M0:
   print("user connection successful")
s = str(sk)+str(k1)
M2 = NewID ^ int(hashlib.sha256(s.encode('utf-8')).hexdigest(),16)

s = str(hashlib.sha256(s1.encode('utf-8')).hexdigest())+str(M3)+str(M4)
M51 = hashlib.sha256(s.encode('utf-8')).hexdigest()
if M5 != M51:
   print("Error")
s = str(did[a])+str(k7)
didnew[a]= M3 ^ int(hashlib.sha256(s.encode('utf-8')).hexdigest(),16)
s = str(f[a])+str(k1)
fnew[a] = M4 ^ int(hashlib.sha256(s.encode('utf-8')).hexdigest(),16)
s = str(NewPW) +str(Newomega)
Cnew[a] = fnew ^ int(hashlib.sha256(s.encode('utf-8')).hexdigest(),16)
did[a] = didnew[a]
Cnew[a] = C[a]
```

CODE(Current Scheme):

initalization.py:

```
import random,nacl,math,binascii


def strtoint(x):
  e = x.encode('utf-8')
  h = binascii.hexlify(e)
  i = int(h, 16)
  return i

def strxor(a, b):     # xor two strings of different lengths
    if len(a) > len(b):
        return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a[:len(b)], b)])
    else:
        return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a, b[:len(a)])])

XG = "qwertyuiop"
x = random.randrange(0,99999,1)
phi = random.randrange(0,99999,1)
sid = random.randrange(0,99999,1)

def is_prime(n):
  if n == 2 or n == 3: return True
  if n < 2 or n%2 == 0: return False
  if n < 9: return True
  if n%3 == 0: return False
  r = int(n**0.5)
  f = 5
  while f <= r:
    if n%f == 0: return False
    if n%(f+2) == 0: return False
    f +=6
  return True

primes = [i for i in range(999,9999) if is_prime(i)]
p = random.choice(primes)

def chebyshev(n,x,p):
    if n==0:
```

```
        return 0
    elif n==1:
        return (x%p)
    elif n>=2:
        return (((2*x*chebyshev(n-1,x,p))-(chebyshev(n-2,x,p)))%p)

PG = chebyshev(phi,x,p)

S = []
S.append(sid)

e = nacl.hash.sha256(str(sid)+str(XG))
```

registration.py:

```
from intialization import XG,strxor
import nacl,random
from fuzzy_extractor import FuzzyExtractor


print("Please enter your ID")
ID = input()
f = nacl.hash.sha256(str(ID)+str(XG))

print("Enter your password")
PW = input()

print("Imprint your thumb")
B = random.randrange(1000,9999,1)

extractor = FuzzyExtractor(16, 8)
omega, theta = extractor.generate(bin(B)) # key, helper
gamma = random.randrange(256,1024,1)
A = (nacl.hash.sha256(str(ID)+str(PW)+str(omega))%gamma)
C = strxor((f) , (nacl.hash.sha256(str(PW)+str(omega))))
```

authentication.py:

```
import random,nacl,sys,time
from registration import B, gamma, theta, A, omega, C, ID
from fuzzy_extractor import FuzzyExtractor
from intialization import chebyshev, x, p, PG, sid,strxor


print("Enter your ID")
```

```
ID1 = input()
print("Enter your password")
PW1 = input()
print("Imprint your thumb")
B1 = B
extractor = FuzzyExtractor(16, 8)
omega1 = extractor.reproduce(bin(B1), theta)
A1 = (nacl.hash.sha256(str(ID1)+str(PW1)+str(omega1)))%(gamma)

temp = 0
while temp!=1:
    if (A) == (A1):
        temp = 1
        continue
    elif (A) != (A1) :
        print("Authentication failed. Please Try Again")
        temp = 1
        sys.exit()
f = strxor(C , nacl.hash.sha256(str(PW1)+str(omega1)))
alpha = random.randrange(0,99999,1)
E = chebyshev(alpha,x,p)
did = str((chebyshev(alpha,PG,p)) ^ (ID))
T1 = time.time()
L = strxor((sid) , (nacl.hash.sha256(str(f)+str(T1))))
Q = nacl.hash.sha256(str(f)+str(E))
N = nacl.hash.sha256(str(ID)+str(Q)+str(sid)+str(T1))

authentication1.py:
import random,nacl,sys,time
from registration import B, gamma, theta, A, omega, C, ID
from fuzzy_extractor import FuzzyExtractor
from intialization import chebyshev, x, p, PG, sid, phi, XG, strxor
from authentication import did, E, L, N, T1


T1C = time.time()
diff = T1C - T1
temp = 0
while temp!=1:
    if diff <= 600:
        temp = 1
        continue
    elif diff > 600 :
```

```
        print("Session is expired")
        temp = 1
        sys.exit()

ID = str((did) , (chebyshev(phi,E,p)))
f1 = nacl.hash.sha256(str(ID)+str(XG))
sid1 = strxor((L) , (nacl.hash.sha256(str(f1)+str(T1)))
Q1 = nacl.hash.sha256(str(f1)+str(E))
N1 = nacl.hash.sha256(str(ID)+str(Q1)+str(sid1)+str(T1))
temp = 0
while temp!=1:
    if  (N1) ==  (N) :
        temp = 1
        continue
    elif  (N1) !=  (N)
        print("Authentication failed. Please Try Again")
        temp = 1
        sys.exit()
e = nacl.hash.sha256(str(sid1)+str(XG))
Y =  strxor(nacl.hash.sha256(str(e)+str(E))) , Q1)
T2 = time.time()
D = nacl.hash.sha256(str(Q1)+str(Y)+str(e)+str(E)+str(T2))

authentication2.py:
import random,nacl,sys,time
from registration import B, gamma, theta, A, omega, C, ID
from fuzzy_extractor import FuzzyExtractor
from intialization import chebyshev, x, p, PG, sid, phi, XG, strxor
from authentication import did, E, L, N, T1
from authentication1 import D,Y,E,T2,e


T2C = time.time()
diff = T2C - T2
temp = 0
while temp!=1:
    if diff <= 600:
        temp = 1
        continue
    elif diff > 600 :
        print("Session is expired")
        temp = 1
        sys.exit()
```

```
Q2 = strxor((Y , (nacl.hash.sha256(str(e)+str(E))))
D1 = nacl.hash.sha256(str(Q2)+str(Y)+str(e)+str(E)+str(T2))
temp = 0
while temp!=1:
    if (D1) == (D) :
        temp = 1
        continue
    elif (D1) != (D) :
        print("Authentication failed. Please Try Again")
        temp = 1
        sys.exit()
beta = random.randrange(0,99999,1)
R = chebyshev(beta,x,p)
K = chebyshev(beta,E,p)
sk = nacl.hash.sha256(str(K)+str(Q2))
G = nacl.hash.sha256(str(sk)+str(R))
T3 = time.time()
W = nacl.hash.sha256(str(R)+str(G)+str(e)+str(E)+str(T3))
```

authentication3.py:

```
import random,nacl,sys,time
from registration import B, gamma, theta, A, omega, C, ID
from fuzzy_extractor import FuzzyExtractor
from intialization import chebyshev, x, p, PG, sid, phi, XG, strxor
from authentication import did, E, L, N, T1
from authentication1 import D,Y,E,T2,e,f1
from authentication2 import R,G,W,T3,sk


T3C = time.time()
diff = T3C - T3
temp = 0
while temp!=1:
    if diff <= 600:
        temp = 1
        continue
    elif diff > 600 :
        print("Session is expired")
        temp = 1
        sys.exit()
W1 = nacl.hash.sha256(str(R)+str(G)+str(e)+str(E)+str(T3))
temp = 0
while temp!=1:
```

```
    if  (W1) ==  (W) :
        temp = 1
        continue
    elif  (W1) !=  (W) :
        print("Authentication failed. Please Try Again")
        temp = 1
        sys.exit()
T4 = time.time()
M = nacl.hash.sha256(str(R)+str(G)+str(f1)+str(T4))
```

authentication4.py:

```
import random,nacl,sys,time
from registration import B, gamma, theta, A, omega, C, ID
from fuzzy_extractor import FuzzyExtractor
from intialization import chebyshev, x, p, PG, sid, phi, XG, strxor
from authentication import did, E, L, N, T1,f,alpha,Q
from authentication1 import D,Y,E,T2,e
from authentication2 import R,G,W,T3,sk
from authentication3 import R,M,T4,W1


T4C = time.time()
diff = T4C - T4
temp = 0
while temp!=1:
    if diff <= 600:
        temp = 1
        continue
    elif diff > 600 :
        print("Session is expired")
        temp = 1
        sys.exit()
K1 = chebyshev(alpha,R,p)
sk = nacl.hash.sha256(str(K1)+str(Q))
G1 = nacl.hash.sha256(str(sk)+str(R))
M1 = nacl.hash.sha256(str(R)+str(G1)+str(f)+str(T4))
temp = 0
while temp!=1:
    if  (M1) ==  (M) :
        temp = 1
        continue
    elif  (M1) !=  (M) :
        print("Authentication failed. Please Try Again")
```

```
        temp = 1
        sys.exit()

passwordupdate.py:
import random,nacl,sys,time
from registration import B, gamma, theta, A, omega, C, ID
from fuzzy_extractor import FuzzyExtractor
from intialization import chebyshev, x, p, PG, sid, phi, XG, strxor


print("Please enter your ID")
ID1 = input()

print("Enter your password")
PW1 = input()

print("Imprint your thumb")
B1 = B

extractor = FuzzyExtractor(16, 8)
omega1 = extractor.reproduce(bin(B1), theta)
A1 = (nacl.hash.sha256(str(ID1)+str(PW1)+str(omega1))%gamma)
temp = 0
while temp!=1:
    if  (A) == (A1):
        temp = 1
        print("Enter your new password")
        continue
    elif  (A) !=  (A1) :
        print("Authentication failed. Please try again")
        temp = 1
        sys.exit()
PWnew = input()
f = (C) ^  (nacl.hash.sha256(str(PW1)+str(omega1)))
Cnew = (f) ^ (nacl.hash.sha256(str(PWnew)+str(omega1)))
Anew = (nacl.hash.sha256(str(ID1)+str(PWnew)+str(omega1))%gamma)
C = Cnew
A = Anew
```