Secret Sharing based Continuous Authentication Algorithm

# Client-side functions

## 1.1 Polynomial Generator (secret, degree k)

Generate a polynomial of degree k, f(x) = a0 + a1\*x + a2\*x^2 + ... + ak\*x^k and return array a = [a0, a1, …, ak] such that:

1. a0 = secret
2. a1, a2, …, ak = randomly sampled integers between (1, 100)

## 1.2 Share Generator (secret, a, x, time flag)

Return timebound share and share authenticator for random x such as:

1. share (u) = f(x) + time flag
2. share authenticator (sa) = hash (share – secret – time flag)

time flag is initialized to 1 and incremented in each continuous authentication

## 1.3 Message Generator (secret, server id, client id, message, share, timestamp, time flag, share authenticator)

1. Generate mac = HMACsecret (client id, server id, message, share, timestamp, time flag)
2. Message to send = {client id, server id, message, share, timestamp, time flag, share authenticator, mac}

# 2. Server-side functions

## 2.1 Authenticator (secret, received shares, message received, time margin)

1. Check message freshness

Time margin = authentication period \* 0.2

* 1. If current timestamp – received timestamp <= time margin:
     + Message is fresh
  2. Else:
     + Message is stale
     + Authentication failed

1. Check Share freshness (has not been used before in the session)
2. Compute fresh mac and check data integrity
   1. mac = HMACsecret (client id, server id, message, share, timestamp, time flag)
   2. If mac = received mac:
      * Message has not been altered and sender is using valid secret
   3. Else:
      * Authentication failed
3. Check if share is authentic
   1. Compute new share authenticator using received share and known secret
      * Share authenticator = hash (received share – secret – received time flag)
   2. If share authenticator = received share authenticator:
      * Share is authentic
      * Client authentication complete (Authentication pass)
   3. Else:
      * Share is not authentic
      * Authentication failed

# 3. Algorithm

1. Secret, authentication period and total session period are initialized and communicated between client and server
2. During session, for every authentication period:
   1. Client side: Message generation
      1. a = Polynomial Generator (secret, degree)
      2. select random int as x
      3. share, share authenticator = Share Generator (secret, a, x, time flag)
      4. Compute message to send as Message Generator (secret, server id, client id, message, share, timestamp, time flag, share authenticator)
      5. Send message: {client id, server id, message, share, timestamp, time flag, share authenticator, mac}
      6. Increment time flag by 1
   2. Server side: Message authentication
      1. Receive message: {client id, server id, message, share, timestamp, time flag, share authenticator, mac}
      2. Check message authenticity as:

Authentication result = Authenticator (secret, received share, message received, time margin)

* + 1. If Authentication result = Pass:
       1. Reset number of failures to 0
       2. Reset backoff period to 0
       3. Send {Authentication Result, Backoff Period} to client
    2. If Authentication result = Fail:
       1. Increment number of failures by 1
       2. Set back off period = authentication period ^ number of failures
       3. Send {Authentication Result, Backoff Period} to client
       4. Do not receive message from client for backoff period
  1. Client side: Backoff in case of authentication failure
     1. If Authentication result = Fail:
        1. Do not send message for backoff period