I utilized the scripts shown below to accomplish both tasks. The associated text outputs are attached as well.

Shrinking Generator

```
def lfsr(state, taps):
  """
  state - current state, initialized to IV
  taps - tap indices

  """
  sum = 0
  for tap in taps:
    sum += int(state[tap])  # Sum taps
  if sum % 2 == 0.0:
    sum = 0
  else:
    sum = 1

  # Final register contains output bit:
  output = state[-1]

  # Move sum to first register, shift all others over
  state = str( sum ) + state[:-1]

  return output, state


def main():
  # Define variables, and define IV/taps for each LFSR:
  outputs_A = []
  states_A, states_S = [], []
  IV_A = '010101'
  IV_S = '11100'

  # Taps are indexed from left to right:
  taps_A = (0, 5)
  taps_S = (1, 4)

  # Clock both LFSRs from their IV:
  outBit_A, stateOut = lfsr(IV_A, taps_A )
  states_A.append( stateOut )

  outBit_S, stateOut = lfsr(IV_S, taps_S )
  states_S.append( stateOut )
  # Check if first bit is counted:
  if outBit_S == '1':
    outputs_A.append( outBitA )

  while len(outputs_A) < 100:
    # Run S-LFSR:
    outBit_S, stateOut_S = lfsr(states_S[-1], taps_S)
    states_S.append(stateOut_S)

    # Run A-LFSR
    outBit_A, stateOut_A = lfsr(states_A[-1], taps_A)
```

```
    states_A.append(stateOut_A)

    if outBit_S == '1':
      outputs_A.append( outBit_A )


  # Output vector to file:
  outFile = open("ShrinkingGen_Output.txt", "w")
  outFile.write("[" + ", ".join( outputs_A ) + "]")
  outFile.close()


if __name__ == "__main__":
  main()
```

ShrinkingGenerator.py

```
[1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0,
 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0,
 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0,
 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1]
```

ShrinkingGen_Output.txt

Feistel Network

The below python script was utilized to perform the Feistel network function.

```
def getLeft( input ):
  """
  Left input is the lower order nibble of the input byte:
  """
  return input & 15


def getRight( input ):
  """
  Right input is the upper order nibble of the input byte:
  """
  # Right shift by 4 will return desired bits by removing the lower order bits
  return input >> 4


def buildByte(right, left):
  """
  This function multiplies the right by 16 to return it to its higher order position
  """
  right = right * 16
  return right + left


def Feistel(inByte):
  """
  inByte - input data
  outByte - taken from the final round; left/right outputs are the would-be left/
    right inputs to the next round.
  """
  # S Boxes for each round:
  S = [ [5, 2, 15, 10, 6, 13, 7, 4, 14, 0, 1, 3, 12, 8, 9, 11],\
  [15, 5, 13, 11, 0, 12, 4, 8, 14, 10, 9, 2, 6, 1, 7, 3],\
  [7, 1, 2, 12, 0, 5, 9, 8, 14, 13, 15, 4, 10, 11, 3, 6] ]

  # Round Keys:
  keys = [15, 5, 9]

  # Pull nibbles from input byte:
  left, right = getLeft(inByte), getRight(inByte)

  for round in range(3):
    # xor right and round key:
    fOut = keys[round] ^ right

    # substitute the output of the fOut with the round's S-Box
    sOut = S[round][fOut]

    # Set next round's inputs:
    rightOut = left ^ sOut
    leftOut = right

    # Reset inputs to next state:
    left = leftOut
```

```
    right = rightOut

  # Build output:
  outByte = buildByte(right, left)
  return outByte


def main():
  # print( lfsr('100100', (5, 0), 128) )
  bytes = [0xFA, 0xB1, 0x39, 0x45]
  outputs = []
  for byte in bytes:
    outputs.append( hex( Feistel( byte ) ) )

  # Print outputs:
  print(outputs)

  # Write outputs to file keeping array syntax:
  outFile = open("FeistelOut.txt", "w")
  outFile.write("[" + ", ".join(outputs) + "]")
  outFile.close()

  print("Done!")

  """ Function testing:
  # Used to test getLeft, getRight, and buildByte
  for byte in bytes:
    left = getLeft(byte)
    right = getRight(byte)
    print( "Left: " + str( getLeft(byte) ) )
    print( "Right: " + str( getRight(byte) ) )
    print( "Original: " + hex( buildByte(right, left) ) )
  """


if __name__ == "__main__":
  main()
```

Feistel.py

```
[0x96, 0x16, 0xc, 0xff]
```

FeistelOut.txt