Algorithm 1 The Lyra2 Algorithm, with p parallel instances.

PARAM: H ▷ Sponge with block size b (in bits) and underlying permutation f
PARAM: ρ ▷ Number of rounds of f during the Setup and Wandering phases
PARAM: Rt ▷ Number of bits to be used in rotations (recommended: a multiple of the machine's word size, W)
PARAM: p ▷ Degree of parallelism ($p \geqslant 1$ and $p|(R/2)$)
PARAM: σ ▷ Number of syncronizes
INPUT: pwd ▷ The password
INPUT: $salt$ ▷ A salt
INPUT: T ▷ Time cost, in number of iterations
INPUT: R ▷ Number of rows in the memory matrix
INPUT: C ▷ Number of columns in the memory matrix (recommended: $C \cdot \rho \geqslant \rho_{max}$)
INPUT: k ▷ The desired key length, in bits
OUTPUT: K ▷ The password-derived k-long key

for each i in $[0, p[$ **do** ▷ Operation performed in parallel, by each thread
  ▷ BOOTSTRAPPING PHASE: Initializes the sponge's state and local variables
  $params \leftarrow len(k) \parallel len(pwd) \parallel len(salt) \parallel T \parallel R \parallel C$ ▷ Multi-byte representation of input parameters (others can be added)
  $H_i.absorb(\mathtt{pad}(pwd \parallel salt \parallel params))$ ▷ Padding rule: 10*1. Password can be overwritten from this point on
  $H_i.absorb(Int(p, b/2) \parallel Int(i, b/2))$ ▷ Absorbs extra block with parallelism configuration
  $prev^0 \leftarrow 2$; $row^1 \leftarrow 1$; $prev^1 \leftarrow 0$
  $gap \leftarrow 1$; $stp \leftarrow 1$; $wnd \leftarrow 2$; $j \leftarrow i$; $sync \leftarrow 1$
  ▷ SETUP PHASE: Initializes a $(R \times C)$ memory matrix, it's cells having b bits each
  for ($col \leftarrow 0$ **to** $C-1$) **do** $\{M_i[0][C-1-col] \leftarrow H_i.squeeze_\rho(b)\}$ **end for** ▷ Initializes $M[0]$
  for ($col \leftarrow 0$ **to** $C-1$) **do** $\{M_i[1][C-1-col] \leftarrow M_i[0][col] \oplus H_i.duplex_\rho(M_i[0][col], b)\}$ **end for** ▷ Initializes $M[1]$
  for ($col \leftarrow 0$ **to** $C-1$) **do** ▷ Initializes $M[2]$ and updates $M[0]$
    $rand \leftarrow H_i.duplex_\rho(M_i[0][col] \boxplus M_i[1][col])$
    $M_i[2][C-1-col] \leftarrow M_i[1][col] \oplus rand$
    $M_i[0][col] \leftarrow M_i[0][col] \oplus rotRt(rand)$ ▷ $rotRt()$: right rotation by L bits (e.g., 1 or more words)
  end for
  for ($row^0 \leftarrow 3$ **to** $R/p-1$) **do** ▷ **Filling Loop**: initializes remainder rows
    for ($col \leftarrow 0$ **to** $C-1$) **do**
      $rand \leftarrow H_i.duplex_\rho(M_i[prev^0][col] \boxplus M_j[prev^1][col] \boxplus M_j[row^1][col], b)$
      $M_i[row^0][C-1-col] \leftarrow M_i[prev^0][col] \oplus rand$
      $M_j[row^1][col] \leftarrow M_j[row^1][col] \oplus rotRt(rand)$
    end for
    $prev^0 \leftarrow row^0$; $prev^1 \leftarrow row^1$; $row^1 \leftarrow (row^1 + stp) \bmod wnd$
    if ($row^1 = 0$) **then** $\{stp \leftarrow wnd + gap$; $wnd \leftarrow wnd \cdot 2$; $gap \leftarrow -gap$; $SyncronizeThreads\}$ **end if**
    if ($row^0 \geq sync \cdot (R/(\sigma \cdot p)) - 1$) **then** $\{sync \leftarrow sync + 1$; $j \leftarrow (j+1) \bmod p$; $SyncronizeThreads\}$ **end if**
  end for
  $SyncronizeThreads$
  ▷ WANDERING PHASE: Iteratively overwrites (random) cells of the memory matrix
  $wnd \leftarrow R/2p$; $sync \leftarrow 1$; $prev^0 \leftarrow wnd - 1$
  $\mathtt{sideA} \leftarrow sync \bmod 2$; $\mathtt{sideB} \leftarrow (sync + 1) \bmod 2$
  for ($\tau \leftarrow 1$ **to** T) **do** ▷ **Time Loop**
    for ($rowCont \leftarrow 0$ **to** $R/p-1$) **do**
      $row^0 \leftarrow \mathrm{LSW}(rand) \bmod wnd$; $row^0_p \leftarrow \mathrm{LSW}(rotRt(rand)) \bmod wnd$
      $j \leftarrow \mathrm{LSW}(rotRt^2(rand)) \bmod p$
      for ($col \leftarrow 0$ **to** $C-1$) **do**
        $col^0 \leftarrow \mathrm{LSW}(rotRt^3(rand)) \bmod C$
        $TempSum \leftarrow TempSum \boxplus M_i[row^0 + (wnd \cdot \mathtt{sideA})][col] \boxplus M_i[prev^0 + (wnd \cdot \mathtt{sideA})][col^0]$
        $TempSum \leftarrow TempSum \boxplus M_j[row^0_p + (wnd \cdot \mathtt{sideB})][col]$
        $rand \leftarrow H_i.duplex_\rho(TempSum)$
        $M_i[row^0 + (wnd \cdot \mathtt{sideA})][col] \leftarrow M_i[row^0 + (wnd \cdot \mathtt{sideA})][col] \oplus rand$
      end for
      $prev^0 \leftarrow row^0$
      if ($rowCont \geq sync \cdot (R/(\sigma \cdot p)) - 1$) **then**
        $sync \leftarrow sync + 1$; $\mathtt{sideA} \leftarrow sync \bmod 2$; $\mathtt{sideB} \leftarrow (sync + 1) \bmod 2$; $SyncronizeThreads$
      end if
    end for
    $SyncronizeThreads$
  end for
  ▷ **Wrap-up phase**: key computation
  $H_i.absorb(M_i[row^0][col^0])$ ▷ Absorbs a final column with the full-round sponge
  $K_i \leftarrow H_i.squeeze(k)$ ▷ Squeezes k bits with a full-round sponge
end for ▷ All threads finished
return $K_0 \oplus \ldots \oplus K_{p-1}$ ▷ Provides k-long bitstring as output