Algorithm 2 The Lyra2 Algorithm.

PARAM: H ▷ Sponge with block size b (in bits) and underlying permutation f
PARAM: H_ρ ▷ Reduced-round sponge for use in the Setup and Wandering phases (e.g., f with ρ)
PARAM: Rt ▷ Number of bits to be used in rotations (recommended: a multiple of the machine's word size, W)
INPUT: pwd ▷ The password
INPUT: $salt$ ▷ A salt
INPUT: T ▷ Time cost, in number of iterations ($T \geqslant 1$)
INPUT: R ▷ Number of rows in the memory matrix (recommended: a power of two)
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

▷ BOOTSTRAPPING PHASE: Initializes the sponge's state and local variables
$params \leftarrow len(k) \,\|\, len(pwd) \,\|\, len(salt) \,\|\, T \,\|\, R \,\|\, C$ ▷ Byte representation of input parameters (others can be added)
$H.absorb(\texttt{pad}(pwd \,\|\, salt \,\|\, params))$ ▷ Padding rule: 10^*1. Password can be overwritten after this point
$gap \leftarrow 1$; $stp \leftarrow 1$; $wnd \leftarrow 2$ ▷ Initializes visitation step and window
$prev^0 \leftarrow 2$; $row^1 \leftarrow 1$; $prev^1 \leftarrow 0$

▷ SETUP PHASE: Initializes a $(R \times C)$ memory matrix, it's cells having b bits each
for $(col \leftarrow 0$ **to** $C-1)$ **do** $\{M[0][C-1-col] \leftarrow H_\rho.squeeze(b)\}$ **end for** ▷ Initializes $M[0]$
for $(col \leftarrow 0$ **to** $C-1)$ **do** $\{M[1][C-1-col] \leftarrow M[0][col] \oplus H_\rho.duplex(M[0][col], b)\}$ **end for** ▷ Initializes $M[1]$
for $(col \leftarrow 0$ **to** $C-1)$ **do** ▷ Initializes $M[2]$ and updates $M[0]$
    $rand \leftarrow H_\rho.duplex(M[0][col] \boxplus M[1][col])$
    $M[2][C-1-col] \leftarrow M[1][col] \oplus rand$
    $M[0][col] \leftarrow M[0][col] \oplus rotRt(rand)$ ▷ $rotRt()$: right rotation by L bits (e.g., 1 or more words)
end for
for $(row^0 \leftarrow 3$ **to** $R-1)$ **do** ▷ **Filling Loop**: initializes remainder rows
    for $(col \leftarrow 0$ **to** $C-1)$ **do** ▷ **Columns Loop**: $M[row^0]$ is initialized, while $M[row^1]$ is updated
        $rand \leftarrow H_\rho.duplex(M[row^1][col] \boxplus M[prev^0][col] \boxplus M[prev^1][col], b)$
        $M[row^0][C-1-col] \leftarrow M[prev^0][col] \oplus rand$
        $M[row^1][col] \leftarrow M[row^1][col] \oplus rotRt(rand)$
    end for
    $prev^0 \leftarrow row^0$; $prev^1 \leftarrow row^1$; $row^1 \leftarrow (row^1 + stp) \bmod wnd$ ▷ Picks rows to be revisited in next loop
    if $(row^1 = 0)$ **then** ▷ Window fully revisited
        $stp \leftarrow wnd + gap$; $wnd \leftarrow 2 \cdot wnd$; $gap \leftarrow -gap$ ▷ Doubles window size and roughly doubles step
    end if
end for

▷ WANDERING PHASE: Iteratively overwrites pseudorandom cells of the memory matrix
for $(\tau \leftarrow 1$ **to** $T)$ **do** ▷ **Time Loop**
    for $(i \leftarrow 0$ **to** $R-1)$ **do** ▷ **Visitation Loop**: $2R$ rows revisited in pseudorandom fashion
        for $(d \leftarrow 0$ **to** $1)$ **do** $\{row^d \leftarrow (\mathrm{LSW}(rotRt^d(rand))) \bmod R\}$ **end for** ▷ Picks pseudorandom rows
        for $(col \leftarrow 0$ **to** $C-1)$ **do** ▷ **Columns Loop**: updates each $M[row^d]$
            for $(d \leftarrow 2$ **to** $3)$ **do** $\{col^d \leftarrow (\mathrm{LSW}(rotRt^d(rand))) \bmod C\}$ **end for** ▷ Picks pseudorandom columns
            $rand \leftarrow H_\rho.duplex(M[row^0][col] \boxplus M[row^1][col] \boxplus M[prev^0][col^0] \boxplus M[prev^1][col^1], b)$
            for $(d \leftarrow 0$ **to** $1)$ **do**
                $M[row^d][col] \leftarrow M[row^d][col] \oplus rotRt^d(rand)$ ▷ Updates the d pseudorandom rows
            end for
        end for ▷ End of Columns Loop
        for $(d \leftarrow 0$ **to** $1)$ **do** $\{prev^d \leftarrow row^d\}$ **end for** ▷ Next iteration revisits most recently updated rows
    end for ▷ End of Visitation Loop
end for ▷ End of the Time Loop

▷ WRAP-UP PHASE: key computation
$H.absorb(M[row^0][col^0])$ ▷ Absorbs a final column with the full-round sponge
$K \leftarrow H.squeeze(k)$ ▷ Squeezes k bits with the full-round sponge
return K ▷ Provides k-long bitstring as output