Awesome—here's a tight, production-ready bundle: a minimal **IdentityRegistry** you can deploy on peaq, plus **Hardhat deploy scripts** that mint AXM, wire roles, set the registry, and (optionally) seed allow/deny lists—mirroring your addresses.json pattern.

1) Minimal Identity Registry (DID/KYC hook)

```
// contracts/IdentityRegistry.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import {AccessControl} from "@openzeppelin/contracts/access/AccessControl.sol";

/**
 * @title IdentityRegistry
 * @notice Minimal KYC/DID registry for AXM. Plug real DID later; interface stays stable.
 * Roles:
 *  - DEFAULT_ADMIN_ROLE: grants/revokes roles, can set flags in emergencies
 *  - ATTESTOR_ROLE: can mark addresses verified/unverified (e.g., your KYC backend)
 */
contract IdentityRegistry is AccessControl {
    bytes32 public constant ATTESTOR_ROLE = keccak256("ATTESTOR_ROLE");

    mapping(address => bool) private _verified;

    event Verified(address indexed user, bool status);

    constructor(address admin) {
        require(admin != address(0), "admin is zero");
        _grantRole(DEFAULT_ADMIN_ROLE, admin);
        _grantRole(ATTESTOR_ROLE, admin);
    }

    function isVerified(address user) external view returns (bool) {
        return _verified[user];
    }

    function setVerified(address user, bool status) external onlyRole(ATTESTOR_ROLE) {
        _verified[user] = status;
        emit Verified(user, status);
    }

    function batchSetVerified(address[] calldata users, bool status) external
onlyRole(ATTESTOR_ROLE) {
        for (uint256 i = 0; i < users.length; i++) {
            _verified[users[i]] = status;
            emit Verified(users[i], status);
        }
    }
}
```

2) AXM (your updated token) — reminder of the interface

You already have the peaq-ready AXM with:

- `setIdentityRegistry(address)`
- `setKYCEnforcement(bool)`
- `setAllowlistEnforcement(bool)`
- `setBlocked(address,bool)`
- `setAllowlisted(address,bool)`

No code repeated here to keep things lean.

3) Hardhat project setup (peaq EVM)

Install:

npm i -D hardhat @nomicfoundation/hardhat-toolbox dotenv
npm i @openzeppelin/contracts

hardhat.config.ts (or .js):

```
import { HardhatUserConfig } from "hardhat/config";
import "@nomicfoundation/hardhat-toolbox";
import * as dotenv from "dotenv";
dotenv.config();

const config: HardhatUserConfig = {
  solidity: {
    version: "0.8.20",
    settings: { optimizer: { enabled: true, runs: 500_000 } }
  },
  networks: {
    // Peaq Agung testnet RPCs are published by peaq—drop yours below
    agung: {
      url: process.env.PEAQ_AGUNG_RPC || "https://agung-rpc.example",
      accounts: process.env.DEPLOYER_KEY ? [process.env.DEPLOYER_KEY] : []
    },
    peaq: {
      url: process.env.PEAQ_MAINNET_RPC || "https://peaq-mainnet-rpc.example",
      accounts: process.env.DEPLOYER_KEY ? [process.env.DEPLOYER_KEY] : []
    }
  }
};
export default config;
```

.env (example):

```
DEPLOYER_KEY=0xabc123...     # deployer PK
PEAQ_AGUNG_RPC=https://...   # peaq testnet RPC
PEAQ_MAINNET_RPC=https://...  # peaq mainnet RPC (later)
DISTRIBUTION_VAULT=0xVault... # where 15B AXM mints
ADMIN_ADDRESS=0xAdmin...      # default admin for roles
```

4) Deploy scripts

4.1 helpers/writeAddresses.ts

```
import fs from "fs";
import path from "path";

const ADDR_PATH = path.join(__dirname, "..", "addresses.json");

export function saveAddresses(network: string, payload: Record<string,string>) {
  let json: any = {};
  if (fs.existsSync(ADDR_PATH)) json = JSON.parse(fs.readFileSync(ADDR_PATH,"utf8"));
  json[network] = { ...(json[network]||{}), ...payload };
  fs.writeFileSync(ADDR_PATH, JSON.stringify(json, null, 2));
  console.log(`Updated addresses.json for ${network}`);
}
```

4.2 scripts/01_deploy_identity.ts

```
import { ethers, network } from "hardhat";
import { saveAddresses } from "../helpers/writeAddresses";

async function main() {
  const admin = process.env.ADMIN_ADDRESS!;
  if (!admin) throw new Error("ADMIN_ADDRESS missing");

  const Registry = await ethers.getContractFactory("IdentityRegistry");
  const registry = await Registry.deploy(admin);
  await registry.deployed();

  saveAddresses(network.name, { IdentityRegistry: registry.address });
  console.log("IdentityRegistry:", registry.address);
}

main().catch((e)=>{ console.error(e); process.exit(1); });
```

4.3 scripts/02_deploy_axm.ts

```
import { ethers, network } from "hardhat";
import { saveAddresses } from "../helpers/writeAddresses";
import fs from "fs";

async function main() {
  const vault = process.env.DISTRIBUTION_VAULT!;
  const admin = process.env.ADMIN_ADDRESS!;
  if (!vault) throw new Error("DISTRIBUTION_VAULT missing");
  if (!admin) throw new Error("ADMIN_ADDRESS missing");

  const AXM = await ethers.getContractFactory("AXM");
  const axm = await AXM.deploy(vault, admin);
  await axm.deployed();

  saveAddresses(network.name, { AXM: axm.address });

  // Optional: wire roles to ops multisigs later if you have them.
  // Already granted to admin in constructor.
```

```
  console.log("AXM deployed:", axm.address);
}

main().catch((e)=>{ console.error(e); process.exit(1); });
```

4.4 scripts/03_wire_compliance.ts

```
import { ethers, network } from "hardhat";
import fs from "fs";
import path from "path";
const ADDR_PATH = path.join(__dirname, "..", "addresses.json");

async function main() {
  const json = JSON.parse(fs.readFileSync(ADDR_PATH,"utf8"));
  const net = network.name;
  const axmAddr = json[net]?.AXM;
  const regAddr = json[net]?.IdentityRegistry;
  if (!axmAddr || !regAddr) throw new Error("Missing AXM or IdentityRegistry address");

  const axm = await ethers.getContractAt("AXM", axmAddr);

  // Point token at registry (doesn't enforce yet)
  const tx1 = await axm.setIdentityRegistry(regAddr);
  await tx1.wait();

  // Optional: turn on allowlist now (strict launch)
  // await (await axm.setAllowlistEnforcement(true)).wait();

  // Optional: seed allowlist (example)
  // const seed = ["0x111...", "0x222..."];
  // for (const a of seed) { await (await axm.setAllowlisted(a, true)).wait(); }

  console.log(`AXM wired to IdentityRegistry on ${net}`);
}

main().catch((e)=>{ console.error(e); process.exit(1); });
```

4.5 scripts/04_flip_kyc_when_ready.ts

```
import { ethers, network } from "hardhat";
import fs from "fs";
import path from "path";
const ADDR_PATH = path.join(__dirname, "..", "addresses.json");

async function main() {
  const json = JSON.parse(fs.readFileSync(ADDR_PATH,"utf8"));
  const net = network.name;
  const axmAddr = json[net]?.AXM;
  if (!axmAddr) throw new Error("AXM not found");

  const axm = await ethers.getContractAt("AXM", axmAddr);
  const tx = await axm.setKYCEnforcement(true);
  await tx.wait();
  console.log("KYC enforcement enabled");
}
```

```
main().catch((e)=>{ console.error(e); process.exit(1); });
```

4.6 scripts/05_ops_blocks.ts (emergency examples)

```
import { ethers, network } from "hardhat";
import fs from "fs";
import path from "path";
const ADDR_PATH = path.join(__dirname, "..", "addresses.json");

async function main() {
  const json = JSON.parse(fs.readFileSync(ADDR_PATH,"utf8"));
  const net = network.name;
  const axmAddr = json[net]?.AXM;
  if (!axmAddr) throw new Error("AXM not found");

  const axm = await ethers.getContractAt("AXM", axmAddr);

  // Pause if needed
  // await (await axm.pause()).wait();

  // Block a bad actor
  // await (await axm.setBlocked("0xBad...", true)).wait();

  // Unpause
  // await (await axm.unpause()).wait();
}

main().catch((e)=>{ console.error(e); process.exit(1); });
```

5) Run order (Agung testnet)

```
# 1) Deploy registry
npx hardhat run scripts/01_deploy_identity.ts --network agung

# 2) Deploy AXM (mints 15B to vault; wires roles to ADMIN)
npx hardhat run scripts/02_deploy_axm.ts --network agung

# 3) Wire AXM -> IdentityRegistry (no KYC enforced yet)
npx hardhat run scripts/03_wire_compliance.ts --network agung

# 4) (Optional) Enable allowlist at launch, then flip KYC later
# npx hardhat run scripts/04_flip_kyc_when_ready.ts --network agung
```

6) Quick test snippets (sanity)

Verify transfers work before toggles:

```
// hardhat console
const axm = await ethers.getContractAt("AXM", "<AXM_ADDR>");
(await axm.balanceOf(process.env.DISTRIBUTION_VAULT)).toString();
```

Turn on allowlist, seed a buyer, and test:

```
await axm.setAllowlistEnforcement(true);
await axm.setAllowlisted("<buyer>", true);
// From vault signer:
await axm.transfer("<buyer>", ethers.parseEther("1000"));
```

Turn on KYC and verify address in registry:

```
const reg = await ethers.getContractAt("IdentityRegistry", "<REG_ADDR>");
await reg.setVerified("<buyer>", true);
await axm.setKYCEnforcement(true);
// Transfers now require reg.isVerified(sender) && reg.isVerified(recipient)
```

7) Notes / audit hygiene

- Keep **kycEnforced=false** and **allowlistEnforced=false** in early internal testing.
- When going public (presale/whitelist), enable **allowlistEnforced=true**; flip **kycEnforced=true** only when your Persona → registry sync is live.
- Store **ADMIN**, **PAUSER**, **COMPLIANCE**, **RESCUER** on **separate multisigs** (or at least separate EOAs) for least-privilege ops.
- Keep `rescueTokens` for hygiene; it cannot pull AXM.

If you want this bundled into a **single Replit AI prompt** (create folders, write files, install deps, and run deploy on Agung), I can generate that next.