```
const { expect } = require("chai");
const { ethers } = require("hardhat");

describe("CrossChain Outward Payment Flow (Direct Deploy in Test)",
function () {
  let deployer, payerVASP, payeeVASP;
  let usdc, lpToken, processor, payerCustodial, deployed;

  before(async function () {
    [deployer, payerVASP, payeeVASP] = await ethers.getSigners();
    console.log("⚡ Deploying contracts directly within test...");

    // ✅ Deploy TestUSDC
    const TestUSDC = await ethers.getContractFactory("TestUSDC");
    usdc = await TestUSDC.deploy();
    await usdc.waitForDeployment();
    const usdcAddress = await usdc.getAddress();
    console.log("USDC deployed at:", usdcAddress);

    // ✅ Deploy CustodialWallet for payer
    const CustodialWallet = await
ethers.getContractFactory("CustodialWallet");
    const payerWallet = await CustodialWallet.deploy(usdcAddress,
payerVASP.address, payerVASP.address);
    await payerWallet.waitForDeployment();
    const payerWalletAddr = await payerWallet.getAddress();

    // ✅ Mint USDC
    await usdc.mint(payerWalletAddr, ethers.parseUnits("2000", 18));
    console.log("✅ Minted 2000 USDC to payer wallet:",
payerWalletAddr);

    // ✅ Deploy remaining contracts as in deploy_all.js
    const HTLCVault = await ethers.getContractFactory("HTLCVault");
    const payerVault = await HTLCVault.deploy(usdcAddress);
    await payerVault.waitForDeployment();

    const LPToken = await ethers.getContractFactory("LPToken");
    lpToken = await LPToken.deploy("Liquidity Provider Token",
"LPT", deployer.address);
    await lpToken.waitForDeployment();

    const CrossChainPaymentProcessor = await
ethers.getContractFactory("CrossChainPaymentProcessor");
    processor = await CrossChainPaymentProcessor.deploy(
      payerVASP.address,
      payerWalletAddr,
      await payerVault.getAddress(),
      usdcAddress
    );
    await processor.waitForDeployment();
    const processorAddress = await processor.getAddress();
    await
payerWallet.connect(payerVASP).setAuthorizedProcessor(processorAddre
ss, true);
    console.log("✅ Processor authorized in Payer
CustodialWallet");

    await lpToken.transferOwnership(await processor.getAddress());
    console.log("✅ LPToken ownership transferred to
CrossChainPaymentProcessor");

    // ✅ Store for later use
    deployed = {
      usdc: usdcAddress,
      payerCustodialWallet: payerWalletAddr,
      payerVault: await payerVault.getAddress(),
      lpToken: await lpToken.getAddress(),
      processor: await processor.getAddress(),
      payerVASP: payerVASP.address
    };

    // ✅ Quick code existence check
    const usdcCode = await ethers.provider.getCode(usdcAddress);
    console.log("🔍 USDC Code Length:", usdcCode.length);
    expect(usdcCode).to.not.equal("0x");
  });

  it("should execute outward payment successfully and validate
metadata", async function () {
    // ✅ You can now use deployed.usdc, processor, lpToken
directly
    const paymentAmount = ethers.parseUnits("1000", 18);

    const lpInfo = {
      tokenAddress: deployed.lpToken,
      tokenType: ethers.encodeBytes32String("USDC-LP"),
      provider: ethers.encodeBytes32String("AAVE"),
      conversionRate: ethers.parseUnits("1", 18)
    };

    const riskMetadata = {
      checkNames: [ethers.encodeBytes32String("AML"),
ethers.encodeBytes32String("KYC")],
      outcomes: [ethers.encodeBytes32String("PASS"),
ethers.encodeBytes32String("PASS")],
      metadataHash: ethers.encodeBytes32String("RISK_OK")
    };

    const paymentInstruction = {
      payerVault: deployed.payerVault,
      payerSC: ethers.encodeBytes32String("USDC"),
      paymentAmount,
      payeeVASP: payerVASP.address, // For now use payer as payee
for testing
      payeeChainNetwork: ethers.encodeBytes32String("CHAIN-B"),
      payeeWallet: deployed.payerCustodialWallet,
      payeeSC: ethers.encodeBytes32String("USDC"),
      lpInfo,
      riskMetadata
    };

    // ✅ Call the processor
    await
expect(processor.connect(payerVASP).executeOutwardPayment(paymentIns
truction))
      .to.emit(processor, "OutwardPaymentExecuted");
  });
});
```