How to Integrate Pimlico with Lit Protocol OTP Authentication

This how-to guide will walk you through the steps to integrate Lit Protocol's OTP sign-in with email, SMS, and Whatsapp with a smart account whose user operations are relayed and sponsored by Pimlico.

Lit Protocol is an Authentication solution that lets you create and manage distributed cryptographic key-pairs for condition-based encryption and programmatic signing. A decentralized key management network, Lit can be used in place of centralized key custodians and other key management solutions. For more information on how Lit Protocol works, visittheir documentation page .

Stytch will be used to manage the OTP authentication flow.

Install the required packages

```

npminstall stytch @lit-protocol/pkp-ethers @lit-protocol/lit-auth-client @lit-protocol/auth-helpers @lit-protocol/types @lit-protocol/lit-node-client-nodejs

```

Make an account with Stytch and get the Project ID and Secret

You can sign up for a Stytch accounthere . Once you have an account, you can find your Project ID and Secret in the Stytch Dashboard API Keys page .

Create a Stytch client with your Project ID and Secret

```

conststytchClient=newstytch.Client({ project_id:"project-test-XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX", secret:"secret-test-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX", });

```

Send an OTP to the user's email, SMS, or Whatsapp

email SMS WhatsApp ```

email conststytchResponse=awaitstytchClient.otps.email.loginOrCreate({ email:'"', })

```

### Authenticate the user with the OTP and get a session token

email SMS WhatsApp ```

email constauthResponse=awaitstytchClient.otps.authenticate({ method_id: stytchResponse.email_id, code: otpResponse.code, session_duration_minutes:60*247*, })

constsessionStatus=awaitstytchClient.sessions.authenticate({ session_token: authResponse.session_token, })

```

Get a Lit Relay Server API Key

You can get a Lit Relay Server API Key by filling outthe Lit Protocol team's form

Mint a PKPs through Lit Protocol

```

constlitClient=newLitAuthClient({ litRelayConfig: { relayApiKey:", } });

constsession=litClient.initProvider(ProviderType.StytchOtp, { userId: sessionStatus.session.user_id, appId:"project-test-XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX" })
```

constauthMethod=awaitsession.authenticate({ accessToken: sessionStatus.session_jwt })

awaitsession.mintPKPThroughRelayer(authMethod) constpkps=awaitsession.fetchPKPsThroughRelayer(authMethod)

```

### Generate the Controller Session Signatures

```

constlitNodeClient=newLitNodeClientNodeJs({ litNetwork:'serrano', debug:false, }) awaitlitNodeClient.connect();

constresourceAbilities=[{ resource:newLitActionResource("*"), ability: LitAbility.PKPSigning, },];

constsessionKeyPair=litNodeClient.getSessionKey();

constauthNeededCallback=async(params:AuthCallbackParams)=>{ constresponse=awaitlitNodeClient.signSessionKey({ sessionKey: sessionKeyPair, statement: params.statement, authMethods: [authMethod], pkpPublicKey: pkp[pkp.length-1].publicKey, expiration: params.expiration, resources: params.resources, chainId:1, }); returnresponse.authSig; };

constsessionSigs=awaitlitNodeClient.getSessionSigs({ chain:"ethereum", expiration:newDate(Date.now()+1000*60*60*24*7).toISOString(), resourceAbilityRequests: resourceAbilities, sessionKey: sessionKeyPair, authNeededCallback }).catch((err)=>{ console.log("error while attempting to access session signatures: ", err) throwerr; });

```

### Initialize the PKP Wallet

We will now generate a wallet that can act a regular Ethers.js wallet, but will use the PKPs minted through Lit Protocol to sign transactions under the hood.

```

constpkpWallet=newPKPEthersWallet({ pkpPubKey: pkp[pkp.length-1].publicKey, rpc:"",// e.g. https://rpc.ankr.com/eth_sepolia controllerSessionSigs: sessionSigs });

awaitpkpWallet.init();

```

### Use the PKP Wallet to sign user operations and send them through Pimlico

You can now use thepkpWallet as a regular Ethers.js wallet to sign user operations. To submit a user operation to Pimlico, you can follow the steps tosponsor a user operation with Pimlico's verifying paymaster and/orsubmit a user operation through Pimlico's bundler . If you would like to integrate Lit Protocol with the full flow of generating, signing, and submitting a user operation, you can follow the steps intutorial 1 , replacing the signing step with the PKP wallet and usingpkpWallet.address as the owner address of the smart account.

Modified fromtutorial 1 , an example of how to use the PKP wallet to sign a user operation is shown below:

```

constsignature=awaitpkpWallet.signMessage(ethers.utils.arrayify(awaitentryPoint.getUserOpHash(userOperation)),)

userOperation.signature=signature

```

And an example of how you would generate the initCode for a SimpleAccount using the PKP wallet is shown below:

```

constinitCode=ethers.utils.hexConcat([SIMPLE_ACCOUNT_FACTORY_ADDRESS, simpleAccountFactory.interface.encodeFunctionData("createAccount", [pkpWallet.address,0]),])

```
```