Signing UserOp

UserOp signature binds the userOpHash, preventing any changes to the userOp once the signature is generated. If any of the userOp fields is changed, it will require re-signing. Signature is generated based on the validation scheme used.

To sign the userOp, first calculate the userOpHash and then sign it using the same signer, account was created. you can find more details about calculating userOp Hashhere

Below code uses ECDSA validation module to sign the userOp.

```
let provider =

new

ethers . providers . JsonRpcProvider ( "https://rpc.ankr.com/polygon_mumbai"

) ;

let signer =

new

ethers . Wallet ( "private key" , provider ) ;

function

getUserOpHash ( useOpMinusSignature :

any )

{ console . log ( "useOpMinusSignature" , useOpMinusSignature ) const packedData = ethers . utils . defaultAbiCoder .
encode ( [ "address" , "uint256" , "bytes32" , "bytes32" , "uint256" , "uint256" , "uint256" , "uint256" , "uint256" , "bytes32" , ] ,
[ useOpMinusSignature . sender , useOpMinusSignature . nonce , ethers . utils . keccak256 ( useOpMinusSignature .
initCode ) , ethers . utils . keccak256 ( useOpMinusSignature . callData ) , useOpMinusSignature . callGasLimit ,
useOpMinusSignature . verificationGasLimit , useOpMinusSignature . preVerificationGas , useOpMinusSignature .
maxFeePerGas , useOpMinusSignature . maxPriorityFeePerGas , ethers . utils . keccak256 ( useOpMinusSignature .
paymasterAndData ) , ] ) ;

const enc = ethers . utils . defaultAbiCoder . encode ( [ "bytes32" ,

"address" ,

"uint256" ] , [ ethers . utils . keccak256 ( packedData ) ,

"0x5ff137d4b0fdcd49dca30c7cf57e578a026d2789" ,

80001 ] ) ;

const userOpHash = ethers . utils . keccak256 ( enc ) ; return userOpHash ; }

async

function

signUserOp

( userOp : UserOperation )

{ const userOpHash =

getUserOpHash ( userOp ) ; console . log ( "userOpHash" , userOpHash )

const moduleSig =

await signer . signMessage ( ethers . utils . arrayify ( userOpHash ) ) ; const signatureWithModuleAddress = ethers . utils .
defaultAbiCoder . encode ( [ "bytes" ,

"address" ] , [ moduleSig ,

"0x0000001c5b32F37F5beA87BDD5374eB2aC54eA8e" ] , ) ; return

{
```

... userOp , signature : signatureWithModuleAddress } ;

} Previous Sponsor using ERC20 tokens Next Smart Accounts