Session Keys - [experimental feature]

Amongst the many prospects created by Account Abstraction, we also have the possibility of creating sessions for dApp interactions.

Sessions can be used to send transactions from a dApp on behalf of a user without requiring their confirmation with a wallet.

The user is guaranteed that the dApp can only execute transactions that comply with the policies of the session up until the session expires. Let's go through the steps on how you could enable sessions in your dApps with Argent:

Installation

```

Copy npm install @argent/x-sessions

# or

yarn add @argent/x-sessions

# or

pnpm add @argent/x-sessions

```

Creating a session as a dApp

First, you need to have a deployed account with a valid Signer. This is the account that will authorize the session and interact with the contracts of your dApp.

A session looks like/resembles this:

```

Copy interfacePolicy{ contractAddress:string selector:string }

interfaceRequestSession{ key:string expires:number policies:Policy[] }

```

You can either get that by using the Account and Signer classes provided by starknet.js or by using an injected instance by a wallet.

```

Copy constaccount=window.starknet.account

```

Next, you need to come up with the permissions you would like for your session. You also need to generate the session keypair to which you want to grant these rights.

This example session will allow the dApp to execute an example endpoint on an example contract without asking the user to approve the transaction again. After signing the session, the dApp can execute all transactions listed inpolicies whenever it wants and as many times as it wants.

```

Copy import{ Signer,ec }from"starknet"

// gets signer with random private key you need to store if you want to reuse the session constsessionSigner=newSigner()

constrequestSession:RequestSession={ key:awaitsessionSigner.getPublicKey(),
expires:Math.floor((Date.now()+1000*60*60*24)/1000),// 1 day in seconds policies:[ { contractAddress:"0x...",
selector:"doAction" } ] }

```

Now you can sign the session with the account you have. Depending on how your account works, the user may get asked to sign a message

```

Copy import{ createSession }from"@argent/x-sessions"

// calls account.signMessage internally constsignedSession=awaitcreateSession(requestSession,account)

```

Using established sessions

With your signed session you can now use it with your dApp to do transactions without the user having to approve again.

```

Copy import{ SessionAccount }from"@argent/x-sessions"

constsessionAccount=newSessionAccount( account, account.address, sessionSigner, signedSession )

// this transaction should get executed without the user having to approve again consttx=sessionAccount.execute({ // lets assume this is a erc20 contract contractAddress:"0x...", selector:"transfer", calldata:[ "0x...", // ... ] })

```

You can also use the session when you don't have access to the main account (window.starknet.account). You only need access to thesignedSession object and thesessionSigner .

```

Copy constsessionAccount=newSessionAccount( providerWithCorrectNetwork, "0xaccountAddress", sessionSigner, signedSession )

```

Having done these, congratulations! you just enabled sessions in your dApp!

NB: It's worth noting that session keys are still an experimental feature and not yet recommended for mainnet apps. However, if you need to use them in your dApp, do reach out to us atdapps@argent.xyz .

Previous Cosigners - Guardians and 2FA Next Paymasters

Last updated3 months ago