Batched Session Validation

This module works hand in hand with the Session Key Manager Module. It allows for multiple session validation modules to be activated for a user.

What it enables

Those who are familiar with the Biconomy Modular Session Keys framework know that it provides great flexibility and allows for quick building of the Session Validation Modules for every new use case without touching the core Session Keys logic. Batched Session Router adds composability, allowing batching of several session key signed operations which should be validated by different Session Validation modules into one User Operation and execute them atomically.

UX Impact

This module allows batching several operations (actions) into one atomically executed User Operation, thus ensuring better UX for many DeFi use cases.

Let’s take a very simple example. Some dApp wants to allow users to perform a simple flow of actions:

Approve token A to a DEX Swap token A for token B Stake token B on some Protocol It also doesn’t want to make this on behalf of the users when the rate of the swap is optimal.

In this case, they will use Session Keys to sign those operations. Since those actions are very common there already are the building blocks for this = appropriate Session Validation Modules (SVMs): ERC20ApprovalSVM, DEXSwapSVM, ProtocolERC20StakeSVM.

Each of those SVMs is only able to validate userOps which specifically performs a given action: swap, approve, or stake. So none of them is able to validate the userOp which leverages executeBatch() method to perform those 3 actions together.

Of course, we can always build 3 separate userOps for those 3 actions, however, they won’t be included in the same bundle as per ERC-4337 specification, so they end up on 3 different bundles, and that’s not what the user wants in DeFi.

It’s also possible to build a custom Session Validation Module that works with this specific flow and validates such atomic userOps which leverages executeBatch(). However, if in the future there’s a need to add one more step to this flow, this will require a new Session Validation Module to be built. Such an SVM would also double-use some of the code already implemented in the basic SVMs and that is not a good practice. Also, this would require permissions to be separately set up for every new SVM.

Session Router addresses those issues by parsing executeBatch() calldata and routing validation flow to the specific SVMs based on the actions into the batch.

Now, dApps can construct flows based on the actions validated by basic SVMs and share common permissions across flows.

How it works

Batched Session Router leveragesSmartAccount.executeBatch() method to execute atomic operations.

It is a Validation module, that validates theuserOps with thecallData field containing a call toSmartAccount.executeBatch() .

Every operation in the batch is expected to be an action managed by a specific Session Validation Module.

Of course, this action should be permitted for a given session key by enabling the appropriate session key + parameters in the Session Key Manager module.

So, the Batched Session Router

1. Verifies every action this userOp claims to perform is enabled for a given Smart Account.
2. Checks which Session Key was used to sign the userOp.
3. Checks this is the actual signer who is allowed to perform the actions.
4. Parses the executeBatch() calldata to get the per-action calldatas and passes them to the appropriate Session Validation Module. SVMs perform permissions checks to ensure that actions that are about to be performed with this calldata comply with the permissions.

It also intersects validity timeframes for all the enabled sessions. if at least one session is expired or not due, the whole atomic operation fails to be validated.

Installation

- npm
- yarn
- pnpm

npm install @biconomy/modules yarn add @biconomy/modules pnpm add @biconomy/modules

Integration

Integration is the same as using the Session Key Manager Module with the additional step of starting a session router module.

import

{ createSessionKeyManagerModule , createBatchedSessionRouterModule ,

DEFAULT_BATCHED_SESSION_ROUTER_MODULE }

from

"@biconomy/account" ;

const sessionModule =

await

createSessionKeyManagerModule ({ moduleAddress :

DEFAULT_SESSION_KEY_MANAGER_MODULE , smartAccountAddress : address , }) ;

const sessionRouterModule =

await

createBatchedSessionRouterModule ({ moduleAddress :

DEFAULT_BATCHED_SESSION_ROUTER_MODULE , sessionKeyManagerModule : sessionModule , smartAccountAddress : saAddress , }) ; Implementation of this is similar to the Session Key Manager module guide - a full implementation example is shownhere Previous Session Key Manager Next Dashboard