# MEV Supply Chain Interface

The previous pages have illustrated how to:

- Build blocks for other chains
- Use confidential computation in various different ways depending on your use case
- Leverage confidential data storage to create a service that is currently run in a centralized way on a decentralized and permissionless network.

Taken together, these three broad categories of the unique SUAVE features allow you to interact with any component in the MEV supply chain. The MEV supply chain looks something like this:

### Off-SUAVE interface

function

submitBundleJsonRPC ( string

memory url ,

string

memory method ,

bytes

memory params )

internal

view

returns

( bytes

memory ) function

submitEthBlockToRelay ( string

memory relayUrl ,

bytes

memory builderBid )

internal

view

returns

( bytes

memory ) function

doHTTPRequest ( HttpRequest memory request )

internal

view

returns

( bytes

memory ) We've already seen in the previous tutorial, which looked at the Private OFA Suapp example contract, how to create bundles and send them to a predefined block builder using thesubmitBundleJsonRPC() precompile.

Similarly speaking, you can construct the whole block yourself and send it to a builder usingsubmitEthBlockToRelay() . Taken together, these two precompiles allow you to interface with thetransaction ,bundle , andblock aspects of the supply chain above.

Furthermore, there is adoHTTPRequest() precompile, which enables Kettles to make any arbitrary http request in order to fetch information from other services they're running on different chains, or just about any other API they need to handle their off chain computation. An example of this in use can also be found in the Private OFA Suapp Example, and it looks like this:

function

submitBundle ( string

memory builderUrl ,

bytes

memory bundleData )

internal

view

returns

( bytes

memory )

{ // encode the jsonrpc request in JSON format. bytes

memory body = abi . encodePacked ( '{"jsonrpc":"2.0","method":"mev_sendBundle","params":[' , bundleData ,

'],"id":1}' ) ;

Suave . HttpRequest memory request ; request . url = builderUrl ; request . method =

"POST" ; request . body = body ; request . headers =

new

string [ ] ( 1 ) ; request . headers [ 0 ]

=

"Content-Type: application/json" ; request . withFlashbotsSignature =

true ;

return Suave . doHTTPRequest ( request ) ; }

## On-SUAVE interface

function

confidentialStore ( DataId dataId ,

string

memory key ,

bytes

memory value )

internal

view function

confidentialRetrieve ( DataId dataId ,

string

memory key )

internal

view

returns

( bytes

memory ) The Private OFA examples also illustrates how the confidential store, and the various on chain interactions in enables, may be used to interface withevery aspect of the supply chain, right from user intents to bundles and blocks.

The below visualization demonstrate the generalized form such Suapps can take as SUAVE matures, illustrating how the OFA contract could key values into the confidential store such that a block building contract could fetch them and use them to create and emit valid blocks (this would require, among other things, extending theallowedPeekers we set inthe previous page ): Edit this page Previous Block Building Next Tutorials