I propose that we change how confidential requests result in suave chain state transitions.

Currently the result

of the execution of a confidential request overrides the calldata used in chain transition. I propose that instead users request suave chain transitions via a new library call Suave.emit_chain_sideeffect

, providing it the calldata which should be executed on chain.

Changelog:

- 5th Dec: renamed onchain

to chain_callback

to emit_chain_sideeffect

- 5th Dec: added a function that returns the log to be emitted prepare_chain_sideeffect

. This makes it impossible to confuse the call for being blocking

library Suave { event OnchainSideeffect(/ ... /) function prepare_chain_sideeffect(address to, bytes data) pure returns OnchainSideeffect {} function emit_chain_sideeffect(address to, bytes data) pure { emit prepare_chain_sideeffect(to, data); } }

This change has two important benefits:

1. Explicit, structured way to execute side effects on suave chain from confidential execution. The structure is easy to extend to accommodate future needs, say attestations.
2. We no longer have to limit to a single on chain side effect as with a return statement. Complex applications that require multiple chain state transitions will be much simpler to develop. In particular this allows modular applications that handle their own chain state transitions, even if parts of the code of the application are not known or the developer does not control them.

There are some less important, but still nice benefits:

- You can request state change from a deployment transaction, not possible previously (since constructors have no returns!)
- You can deploy contracts from confidential transactions, since adding the to

address is trivial

- Once we introduce SGX, attestations become transparent as all callback messages can be attested to on the fly. Something similar can be done while using result for the calldata, but it’s not as simple and elegant because of the need for aggregation of both the results as well as the attestations

.

Usage

contract Suapp { int _remainder function update_rem(int remainder) { _remainder = remainder; }

```
/* New */
function run_confidential(int x) {
    emit Suave.prepare_chain_sideeffect(address(this),
      abi.encodeWithSelector(update_rem.selector, x % 50),
    );
}

/* Old */
function run_confidential(int x) {
    return abi.encodeWithSelector(update_rem.selector, x % 50);
}

}
```

The change seems underwhelming at first, but now imagine that this Suapp is a part of a bigger application. Using Suave.emit_chain_sideeffect

embedding this Suapp is trivial - does not require any changes. Using return to overwrite the calldata embedding even this simple Suapp is already impractical because we have to aggregate the intermediate returns:

contract ParentSuapp { Suapp s

```
function emit_x(int x) {
    emit x;
}

/* New */
function run_confidential(int x) {
    s.run_confidential(x * 17);
    Suave.emit_chain_sideeffect(address(this), abi.encodeWithSelector(emit_x.selector, x));
}

/* Old */
function run_confidential(int x) {
    bytes suappUpdate = s.run_confidential(x * 17);
    return abi.encodeWithSelector(emit_and_run_s.selector, suappUpdate, x);
}
function emit_and_run_s(bytes suappUpdateBytes, int x) {
    address(s).call(suappUpdateBytes);
    emit_x(x);
}

}
```

The Suave.emit_chain_sideeffect

method of requesting chain state transitions is clearly superior in this still trivial example. Developing a large application in the current return-oriented way quickly becomes a UX nightmare.

Specs

For the implementation, I propose that the library call emits an event with the required fields, and the kettle executing the confidential request captures those events (logs) and puts them into the resulting SuaveTransaction

as messages, which are later executed on chain.

Spec PR.