**Advanced EVM - Encoding**

**EVM Encoding**

What we've learnt so far is that any EVM compatible chain is looking for the bytecode of a transaction in order to understand how it's supposed to respond. We've learnt as well that the global functionality of abi.encode, abi.encodePacked and abi.decode can be used to convert almost any data into this bytecode format.

What these two things combined mean is that we can encode our own function calls as data that we send to a contracts address.

If we view a function call on Etherscan, we can see the input data in a human readable form as well as its original form, which is the bytecode representing that function (function selector).

The ability to do this empowers us as developers to do a lot of cool low-level things like making arbitrary function calls.

I've said previously that in order to send a transaction you're always going to need two things:

1. ABI
2. Contract Address

Originally we were referring to the human-readable ABI.

Human-readable ABI

[

{

"inputs": [],

"name": "multiEncode",

"outputs": [

{

"internalType": "bytes",

"name": "",

"type": "bytes"

}

],

"stateMutability": "pure",

"type": "function"

},

{

"inputs": [],

"name": "multiEncodePacked",

"outputs": [

{

"internalType": "bytes",

"name": "",

"type": "bytes"

}

],

"stateMutability": "pure",

"type": "function"

},

{

"inputs": [],

"name": "multiStringCastPacked",

"outputs": [

{

"internalType": "string",

"name": "",

"type": "string"

}

],

"stateMutability": "pure",

"type": "function"

}

]

We can also accomplish our goals with the bytecode version directly. All you *really* need to send a function call is the name of a function and the input types.

Two questions arise:

***How do we send transactions that call functions with just the data field populated?***

***How do we populate the data field?***

We're going to answer these by leveraging additional low-level keywords offered by Solidity, staticcall and call.

We've used call previously... if this code rings a bell:

function withdraw(address recentWinner) public {

(bool success, ) = recentWinner.call{value: address(this).balance}("");

require(success, "Transfer Failed");

}

**call:** How we call functions to change the state of the blockchain

**staticcall:** How we call view or pure functions

❗ **PROTIP** send and delegatecall also exist as options for low-level calling to the blockchain, but we'll go over these in greater detail later!

When we write recentWinner.call{value: address(this).balance}(""); we're directly updating the value property of the transaction we're sending. The parenthesis at the end of this call are where we provide our transaction data.

* within {} we're able to pass specific fields of a transaction, like value
* within () we can pass the data needed to call a specific function.

**Wrap Up**

Whew, this is heavy, but it's advanced. The power provided by low-level function calls cannot be overstated.

Now that we have some understanding of how encoding can be using in sending transactions, let's take a step back in the next lesson to recap what we've gone over