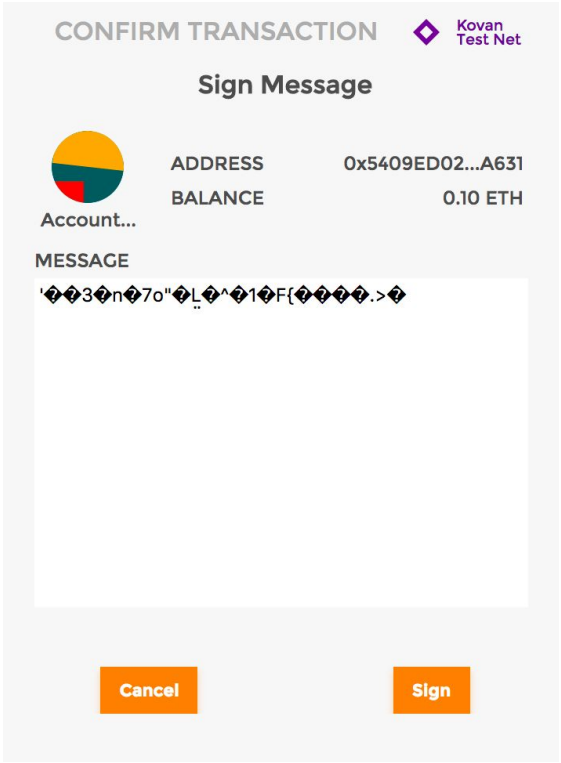
ARAGON


CONFIRM TRANSACTION
Kovan
Test Net
Sign Message
Account...
ADDRESS
0x5409ED02...A631
BALANCE
0.10 ETH
MESSAGE
'��3�n�7o"�L̤�^�1�F{����.>�
Cancel
Sign

Radspec is a **safe** alternative to Ethereum's natspec

```
/**
* @notice Burn `_amount / 10^18` tokens from `_holder`
* @param _holder Holder being removed tokens
* @param _amount Number of tokens being burned
*/
function burn(address _holder, uint256 _amount) authP(BURN_ROLE, arr(_holder, _amount)) isInitialized external {
    // minime.destroyTokens() never returns false, only reverts on failure
    token.destroyTokens(_holder, _amount);
}
```

Demo

Features

- **External calls:** Can perform calls to external contracts
- **Safe:** No DOM access at all
- **Simple:** Very familiar syntax (looks like Flow)
- **Compatible:** Most natspec comments that already exist are also compatible with Radspec

Next steps

- **Frame**: Bringing radspec support to
  - Electron/browser/whatever dapps
  - Any signing method (software/Ledger/Trezor)
- **MetaMask? <3**
- **Some mobile client?**
- **Threat model**:
  - Make it impossible for devs to scam users?
  - TCR for contracts vetted by auditors?

Next steps

- Any smart contract call should have a human-readable description explaining the user the implications of signing a transaction
- Two paths to achieve to get the description:
  - Curated list of function descriptions
  - Smart contract interface claims
- Previous work: https://github.com/ethereum/EIPs/issues/719

Curated list of action descriptions

```
/**
Contract can be deployed using Nick's method so it can have
the same address in every chain, including testnets.
(See 'Deployment instructions': eips.ethereum.org/EIPS/eip-820

A signer needs to slice the first 4 bytes of the tx data,
check if `curatedActionList.getInterface(sig)` returns something
other than an empty bytes array.
*/


contract CuratedActionList {
    modifier auth {
        // GOVERNANCE!!!!
        _;
    }

    mapping (bytes4 => interface) interfaces;

    function registerInterface(bytes4 sig, bytes interface) auth {
        interfaces[sig] = interface;
    }

    function getInterface(bytes4 sig) public view returns (bytes) {
        return interfaces[sig];
    }
```

Curated list of action descriptions

- Pros:
  - Backward compatible with currently deployed contracts
- Cons:
  - Governance over the list is required (TCR?)

Smart contract interface claims

- Claim an interface using ERC780
- Signers can just check whether the target contract has defined an interface
- Interfaces should live off-chain (IPFS)

Smart contract interface claims

```
/**
By deriving from 'InterfaceClaim' the contract, a contract can claim
its interface on create
*/

contract ERC780 {
    function setSelfClaim(bytes32 key, bytes32 value) public;
    function getClaim(address issuer, address subject, bytes32 key) public constant returns (by
    // ... more functions
}

contract InterfaceClaim {
    bytes32 INTERFACE_KEY = keccak256("ERC_X_INTERFACE_KEY");

    function InterfaceClaim(ERC780 erc780, bytes32 interfaceURI) {
        erc780.setSelfClaim(INTERFACE_KEY, interfaceURI);
    }
}
```

Smart contract interface claims

- Pros
  - Each contract can define a custom interface
  - No need to rely on external factors
  - An interface beyond just function descriptions: (See http://voting.aragonpm.com/artifact.json)
- Cons
  - Contracts need to have logic to claim it

Interface format


```
{
    "format": "radspec-v0.3",
    "functions":
        {
            "0xab27fe41": {
                "sig": "initialize()",
                "notice": "Initializes Voting app with `_token.symbol(): string` for governance,
            },
            "0xcb17db31": {
                "sig": "newVote(bytes,address)",
                "notice": "Create a new vote about \"`_metadata`\""
            },
            "0x8ad1efde": {
                "sig": "vote(uint256,bool,bool)",
                "notice": "Vote `_supports ? 'yay' : 'nay'` in vote #`_voteId`"
            }
        }
    }
}
```

Separation of concerns

- **Dapps**: Anything that needs to consume state and produce transactions. Can be running on a browser, Electron, mobile…
- **Signing Providers**: Let users sign their transactions with any provider. Like hardware wallets, software wallets…
- **Web3 Providers**: Let dapps consume state, and let signing providers broadcast transactions

Wallets = kind of dapp

- Let's **not** build **ad-hoc**, **hardcoded** systems!
- Multi-sig, two factor auth, etc. can be used for **much more than moving funds** around

Personal DAO

- Represents you + supports ENS
- Can have multiple apps (vault, fund recovery)
- Can have rich permissions between apps
- Multi-sig, two factor auth, etc. can be used for **much more than moving funds** around

Personal DAO: Example 1

A **Key Split** app that gives **permission** to a set of people to **progressively** access your funds if:

a) You haven't transacted for a month

b) People can send valid signatures to the app

Personal DAO: Example 2

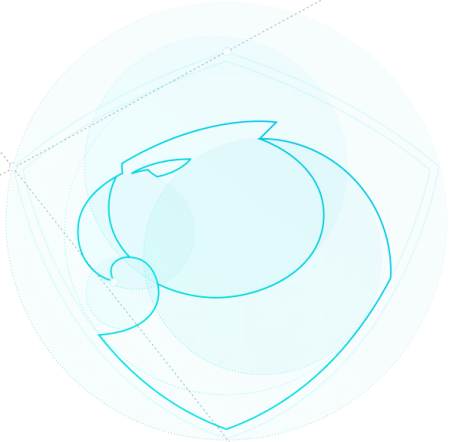
A **Finance** app that gives **permission** to a set of keys to **progressively** access your funds. Example:

- Key A (MetaMask) can access up to 1ETH/month.
- Key A + Key B (MetaMask + Parity Signer) can access up to 10ETH/month.
- Key A + Key B + Key C (MetaMask + Parity Signer + Ledger) can access all of your funds and change permissions in the DAO.

Personal DAO: Example 3

An **Identity** app that can forward posts to Leeroy or Peepeth on your behalf.
You can **re-use any governance mechanism** and give it **permission** over the Identity app.

Organization-controlled social media accounts. Boom!



app.aragon.one

github.com/aragon/radspec