Wrapped DAI

By Alan Lai, Phillip Liu, Leland Lee

The Problem - Terrible User Experience

When a user only has tokens such as DAI in their wallets, they are unable to **send** DAI or **call** smart contracts

The Solution? Wrapped DAI!

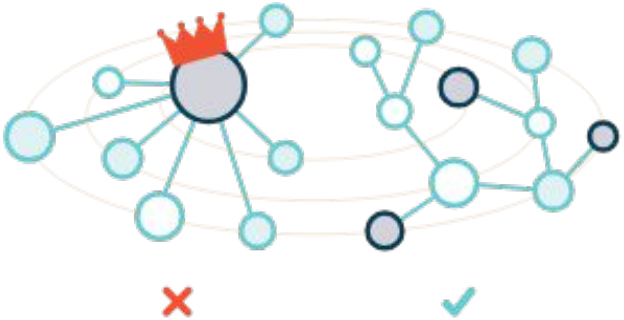
A smart contract that allows individuals with only DAI in their wallets to transact on the Ethereum network. This is done by delegating transaction sending to other network participants and paying them in DAI.

Decentralized Solution while Maintaining Good UX

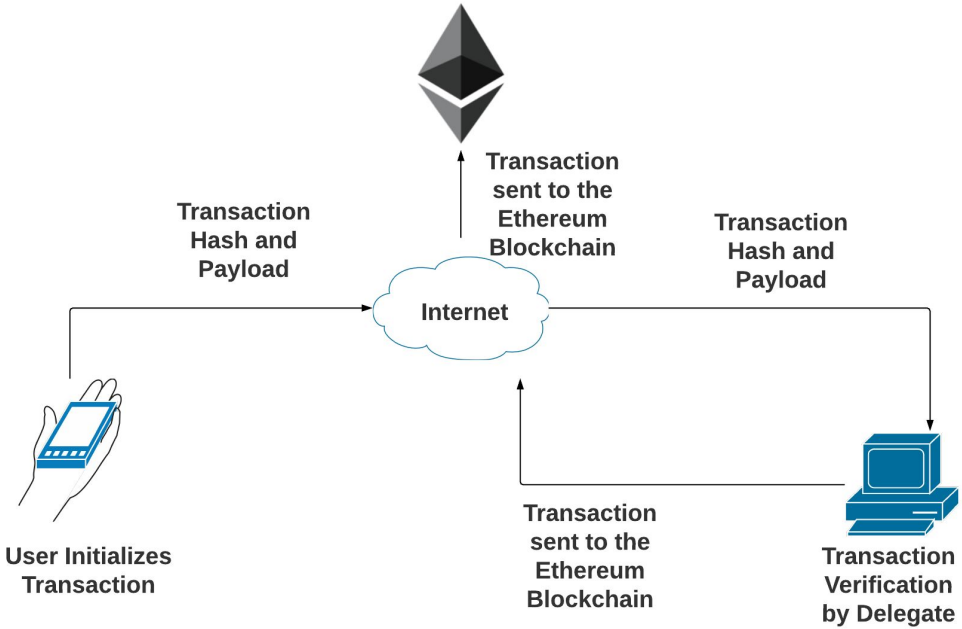
Centralized solutions can solve this issue but they can be easily censored and introduce a single point of failure

What if I don't have access to wyre

Our solution is able to combine the best of both worlds



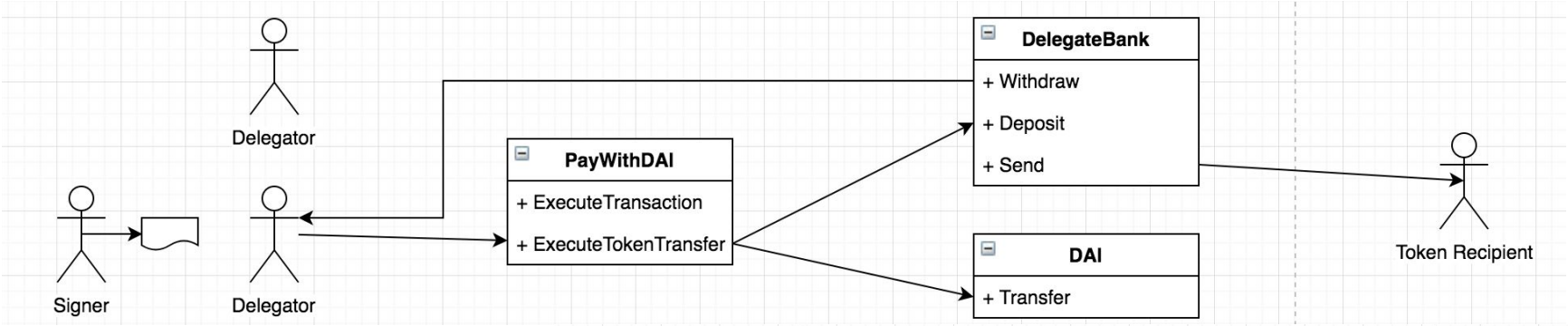
How Does This Work



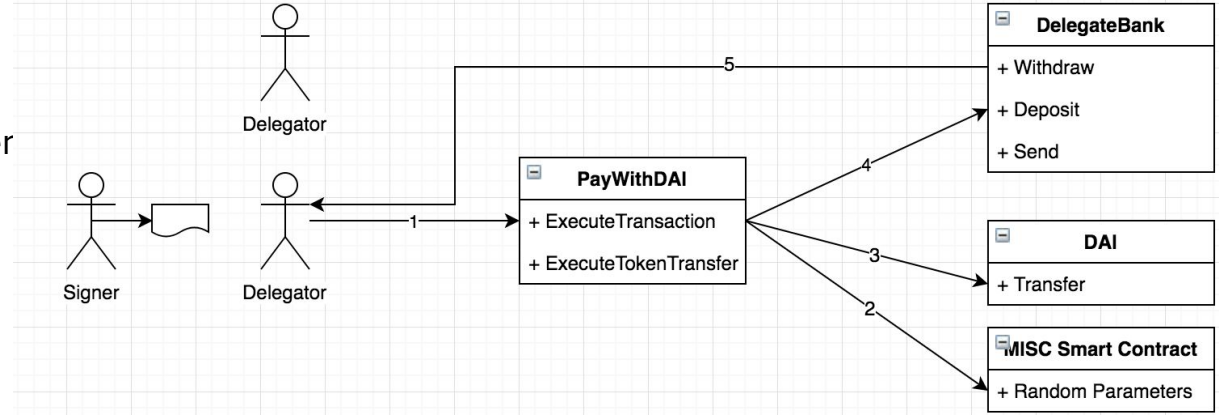
Payload

Method Name	Detail
signer	Address of the signer
Hash	Keccak256 hash of the payload
v,r,s	Recovery id and output of ECDSA signature
fee	Fee paid to Delegator
gasLimit	GasLimit defined by the signer
executeBy	Blockheigh which the Delegator must execute the contract by
executionAddress	Address of smart contract to call
exectuationMessage	message to be passed to the smart contract
feeRecipient	Receiver of the fee

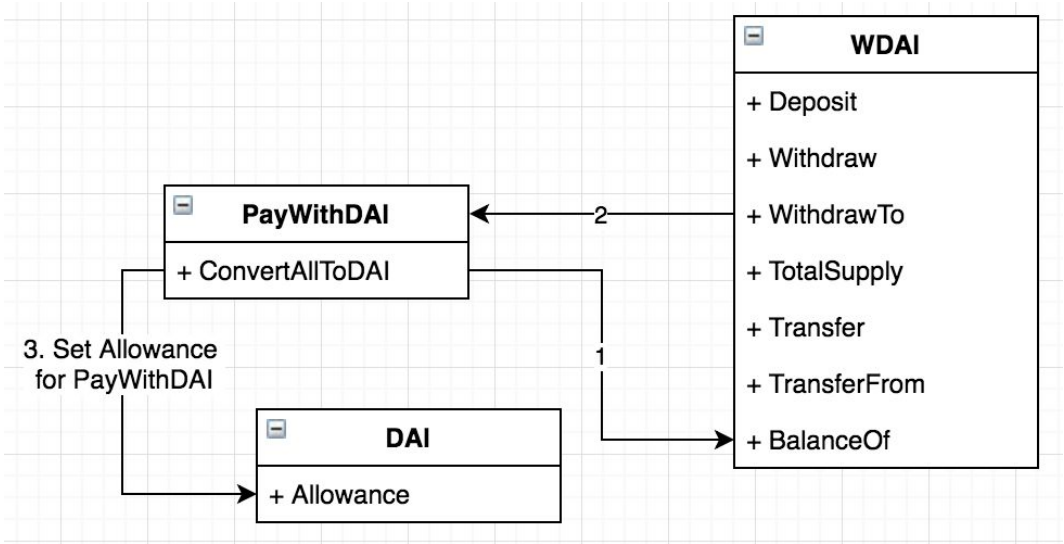
How does this work?



ExecuteTokenTransfer - ERC20 Transfer
ExecuteTransaction - External Smart Contract call (can send ETH)



No Allowances?



In **ERC20** has to set `**allowances**` for other addresses to withdraw from in. An address with only DAI cannot set up this initial allowance. Instead we airdrop people with `**wrapped DAI**` that already has an allowance with the whitelisted `PayWithDAI` contract. As PayWithDAI withdraws on behalf of the signer. Then users' wDAI is now converted to DAI with an allowance with the Whitelisted contract

Features

- Double spend resistant
- wDAI removes `approval` setup step
- Send ERC20s without ETH
- Transact with smart contracts without ETH

Function Interaction Used by Delegator

- verifySignature() checks that signature is from the signer
  - Checks that signature hasn't been executed before
- convertAllToDAI() converts any wDAI balance to only DAI
  - Checks if there is a Balance of wDAI
  - Call withdrawTo() in WDAI contract
    - Check that msg.sender is DelegateBank
    - Clears balance of wDAI
    - Transfers DAI balance of WDAI contract to signer
  - Sets DAI approve for DelegateBank to infinite

Function Interaction Used by Delegator (Cont.)

- verifyPayload() checks that the payload content equals the hash
- verifyExecutionTime() checks if execution height hasn't passed yet
- verifyFunds() checks that signer has enough DAI and appropriate allowance with DelegateBank
- Transfers the amount of DAI required to the DelegateBank
- Calls send() in DelegateBank to transfer funds to intended recipient
- Calls deposit() in DelegateBank to report how much the fee was
- Sets the signature as used, so cannot be replayed
- Calls withdraw() in DelegateBank to withdraw the accumulated fees

DelegatedBank Smart Contract

```
pragma solidity ^0.4.23;

import "./ERC20.sol";
import "./ConvertLib.sol";

contract DelegateBank {

  ERC20 token = ERC20(0xC4375B7De8af5a38a93548eb8453a498222C4fF2);
  address PayWithDAI;
  address owner;
  bool setable;

  mapping (address => uint256) public balances; // Sum of balances

  constructor() public {
    owner = msg.sender;
  }

  function setParent(address parent) public returns(bool) {
    require(msg.sender == owner);
    PayWithDAI = parent;
  }
```

```
// Withdraw is called by Delegators
function withdraw(uint256 amount, address feeRecipient) public returns(bool) {
  require(balances[feeRecipient] >= amount);
  balances[feeRecipient] -= amount;
  /* token.transferFrom(this, feeRecipient, amount); */
  return true;
}

// A deposit function is required as the smart contract is unable to determine
function deposit(uint256 amount, address feeRecipient) public returns(bool) {
  balances[feeRecipient] += amount;
  return true;
}

// Only `PayWithDAI` contract can call this function
function send(address recipient, uint256 amount) public returns(bool) {
  require(msg.sender == PayWithDAI);
  token.transferFrom(this, recipient, amount);
  return true;
}
function getBalance(address addr) public view returns(uint value) {
  return balances[addr];
}
```

PayWithDAI Smart Contract

```
pragma solidity ^0.4.23;

import "./ERC20.sol";
import "./DelegateBank.sol";

contract WDAI {
  function withdrawTo(address signer, uint256 wad) public {}
  function balanceOf(address src) public returns(uint256) {}
}

contract PayWithDAI {
  event ValidSignature(bool validSignature);
  event SufficientFunds(bool sufficientBalance, bool sufficientAllowance);
  event ValidPayload(bool validPayload);
  event TransactionDelegationComplete(address feeRecipient, uint256 fee);

  address public constant DelegateBank = 0x0; // Incorrect address
  ERC20 token = ERC20(0xC4375B7De8af5a38a93548eb8453a498222C4fF2); // DAI ada
  WDAI wtoken = WDAI(0x0); // wDAI address --> how to add a new function to t

  mapping (bytes32 => bool) public signatures; // Prevent transaction replays
```

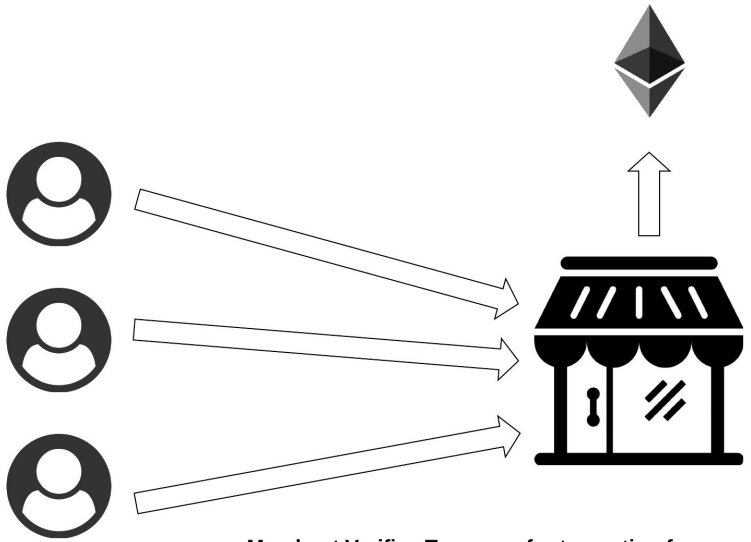
```
function verifySignature(address signer, bytes32 hash, uint8 v, bytes32 r, bytes32 s) private returns(bool) {
  require(signatures[hash] == false);

  bytes memory prefix = "\x19Ethereum Signed Message:\n32";
  bytes32 prefixedHash = keccak256(prefix, hash);
  bool validSignature = ecrecover(prefixedHash, v, r, s) == signer;
  emit ValidSignature(validSignature);

  return validSignature;
}

function verifyPayload(bytes32 hash, uint256 fee, uint256 gasLimit, uint256 executeBy, address executionAddress, bytes32 executionMessage
  bool validPayload = keccak256(abi.encode(fee, gasLimit, executeBy, executionAddress, executionMessage)) == hash;
  emit ValidPayload(validPayload);

  return validPayload;
}

// Note this function is called after `verifySignature` --> signer is known be valid
function verifyFunds(address signer, address feeRecipient, uint256 fee) private returns(bool) {
  bool sufficientBalance = token.balanceOf(signer) >= fee;
  bool sufficientAllowance = token.allowance(signer, feeRecipient) >= fee;
  emit SufficientFunds(sufficientBalance, sufficientAllowance);

  return sufficientBalance && sufficientAllowance;
}
```

PayWithDAI Smart Contract(Cont.)

```
function convertAllToDAI(address signer) public returns(bool) {
  uint256 wtokenBalance = wtoken.balanceOf(signer);
  bool hasWDAI = wtokenBalance > 0;
  if (hasWDAI) {
    wtoken.withdrawTo(signer, wtokenBalance);
    token.approve(DelegateBank, ~uint(0)); // Setting approvals for DAI
  }
  return true;
}

function executeTransaction(address signer, bytes32 hash, uint8 v, bytes32 r, bytes32 s, uint256 fee, uint256 gasLimit, uint256 executeBy
  require(verifySignature(signer, hash, v, r, s));
  require(convertAllToDAI(signer));
  require(verifyPayload(hash, fee, gasLimit, executeBy, executionAddress, executionMessage));
  require(block.number < executeBy); // After payload verification, know executeBy value is correct
  require(verifyFunds(signer, DelegateBank, fee));

  bool executed = executionAddress.call.gas(gasLimit)(executionMessage);

  // What is the difference between putting this into an `if` statement vs `require`
  if(executed) {
    token.transferFrom(signer, DelegateBank, fee); // transfer the tokens from signer -> DelegateBank
    require(DelegateBank.call(bytes4(keccak256("deposit(uint256, address)")), fee, feeRecipient)); // Log the deposit into DelegateBank
    signatures[hash] = true;

    emit TransactionDelegationComplete(feeRecipient, fee);
  }
```

Other use cases - Merchants

Merchants can accept and execute these payments in a peer to peer manner and pay for transaction fees on behalf of customer



Other Use Cases - Airdropped ERC-20

Airdropped ERC-20 tokens can be difficult to move out of wallets without ETH inside already

Our solution can be applied to any ERC-20 token

The Future of Pay With DAI

- Wyre Integration for useable DAI
- Integration with Oasis DEX via Oasis Direct proxy contracts
  - Directly get ETH for trades or for smart contracts
- Enables purchasing of crypto assets such as CryptoKitties (buying items priced in ETH in DAI)
- Implement 0x like relayer
- EIP712 like messages for orders
- Implementing proxy contracts

Contracts

Contract	Address
WDAI	0xd639b18ac72231a38ec219a0f088bbf899eb7533
DelegateBank	0x5f4d4a3c70ea82dceedb3fcfba0064ae34eb6037
PayWithDAI	0xb7c5e4ecc9c16510d2ad9a74943ef784649bef43
Owner	0xE43Ab303b122Ad800aDF5A224b7C3541432CEf61

https://github.com/lelandlee/pay-with-DAI