Prerequisites

- Layer 2 gas payment abstraction
- EIP 1014.

Motivation

Transaction gas abstraction, sometimes known as meta-transactions, should ideally be backward-compatible with existing contracts, most notably ERC20 contracts. We cannot assume that all existing contracts can add @vbuterin's [submitSignedPayments()

](https://ethresear.ch/t/layer-2-gas-payment-abstraction/4513) or any other gas payment abstraction protocol to their code.

On the other hand, we cannot rely on ERC20 approve()

to extend ERC20 contracts in the context of gas payment abstraction because this first step requires the external account to pay for its gas fee in Ether which defies the purpose.

However, I think a solution may be possible in a few days thanks to Constantinople'sEIP 1014.

Basic Protocol

Using EIP 1014

's new CREATE2

opcode we can allow for deployment gas abstraction for any contract, including "smart wallets

", leading to ERC20 backward-compatible layer 2 gas payment abstraction:

- A public factory

contract is deployed once on the mainnet allowing anyone to deploy CREATE2

contracts by providing init_code

and salt

.

- Alice creates a new external account key pair with 0 Ether balance
- Alice generates init_code

for a gas abstraction wallet contract, or smart_wallet

, of her choice and sets a fixed ERC20 fee

, say in Dai, for msg.sender

in the constructor

.

- Alice selects a random salt

and combines it with the init_code

and the factory

address to deterministically generate an address for her smart_wallet

contract before deployment.

- Alice uses her off-chain smart_wallet

address instead of her external account address to receive Dai.

- When the smart_wallet

's Dai balance exceeds the fee

set in the contract constructor

, Alice can send the init_code

and salt

of her smart_wallet

to a relayer, Bob, to deploy it using the factory

and redeem the Dai fee.

- Alice can now use a gas abstraction protocol set in her smart_wallet

.

- Nowhere in this process Alice needs to get or know about Ether.

Discussion

- Alice may receive tokens at her smart_wallet

address but cannot find a relayer potentially because the ERC20 token fee previously set in the constructor

is not, or no longer is, sufficient to cover a relayer's gas fees. The worst case scenario here is one where Alice chooses to acquire Ether on her external account and take the role of the relayer herself. To avoid this, this one time fee must be sufficiently high to accommodate for volatility risks. Although either way, Alice's funds are never stuck forever. A potential improvement would be for the contract's constructor

to know the current token price in Ether using [Uniswap's getTokenToEthInputPrice()

exchange interface](https://docs.uniswap.io/smart-contract-integration/solidity#exchange-interface) to dynamically set the constructor

fee but that may put Alice in some volatility risk herself. Although at a fee scale the volatility may be negligible. If not, again, Alice can relay for herself. Better improvements? Edit:

This problem was solved here.

- Relayer front-running should not be usable as an attack vector against Alice but I'm curious to know how it would affect relayers' availability, price competition and decentralization.
- What are other applications of contract deployment gas abstraction other than counterfactual contracts or backwards compatible submitSignedPayments()

?

- What are the potential attack vectors, bad scenarios, black swan scenarios, etc that I haven't considered?