The concept of meta-transactions was first popularized by Alex Van de Sande in EIP-10771 and Austin Griffith with his burner wallet 2. It has become a staple in high profile projects, such as Reddit's Community Points3. However, new proposals such as UNGAS 4 and Oil [5] threaten to break some of its assumptions.

Relay contracts assume they have the ability to catch reversions in their children, which they do by providing their children a fraction of the total gas so that the call's result can be processed. Changing the semantics of CALL

so that it always sends all

the remaining gas would render this technique useless, since any revert would cascade through entire call stack.

In order to resolve this issue, there will either need to be additional modifications to core protocol or the gas observability proposals will need to properly support meta-transactions. I believe there are two important, but orthogonal attributes that a solution should provide:

1. Atomicity

: it should be possible to construct a protocol which allows for transactions to be paid for using assets other than ether without complex interactions transpiring across multiple blocks.

1. Interoperability

: contracts should not need to provide special interfaces (e.g. EIP-12716 compliant) to authenticate the sender of a message. In other words, CALLER

should return the address of the sender not

the relayer.

Atomicity Solutions

The protocol currently provides atomicity across a single transaction. Atomicity in scenarios where a subcall fails has been achieved using the less-than-best-practice technique described above. If Oil / UNGAS / significant changes in the gas schedule break this functionality, relayers will have to accept the risk that they may pay for a tx without being compensated. Whether or not this is an acceptable risk is being discussed [7]).

Rich Transactions 8

A new reserved address is specified at x, in the range used for precompiles. When a transaction is sent to this address from an externally owned account, the payload of the transaction is treated as EVM bytecode, and executed with the signer of the transaction as the current account.

Pros:

- expressive construction which allows complex scripts to be defined and will open many new use cases
- no modification to transaction object required

Cons:

- if gas becomes completely unobservable, this will not provide atomicity since subcalls will always receive all remaining gas
- large surface area for attacks

Native batched transactions 9

Allow EOA to execute atomic batch transactions, where only one nonce and one signature would be provided for a given batch of transactions.

Pros:

- simple to reason about

Cons:

- transactions are inherently independent of each other, therefore its unclear how pay for the exact amount of gas spent in a previous tx in the batch
- a new transaction object would be needed or the existing one would need to be extended

Account abstraction [10]

Suppose transactions to a reserved address are given a fixed amount of gas and are deemed valid if they execute a new opcode, PAYGAS

, before running out of gas. All other logic, including how to handle the call data is arbitrary – opening the door for “rich transactions”.

Pros

- allows custom transaction validation functions

Cons

- increases the complexity of the tx pool
- does not provide more value than rich transactions
- does not resolve atomicity for EOAs

Transaction-Specifiable Subcall Gas [11]

There are a few mechanisms for achieving this that are enumerated in[11], but the high-level goal is to delocalize the gas provided to subcall so that it can be specified by an external source, like a new field in the transaction object.

Pros

- less complex than some alternatives

Cons

- another quirk added to the EVM
- calls which were not susceptible to reentry may become vulnerable[12]
- gas is still observable [13]

Expansion of Miner Duties [14]

Suppose miners begin to accept requests to relay certain transactions. If they deem a transaction worth relaying (e.g. it will pay them in some other token), then they include it in the block with a gas price of 0.

Pros

- could be implemented without a hard fork
- builds on existing meta-tx work like EIP-1077 1

Cons

- txs paying in esoteric tokens would face increased latency[15]
- the duties of the miner would increase significantly to provide these services[15]
- the relationship is strictly between the signer and the miner, so not all onboarding use cases for dapps would be covered [15]

Do Nothing [16]

So this leads to a philosophical question: why not just implement none of these proposals, keep tweaking gas costs as needed for sustainability as we’ve done before, and just publicly state the social norm that you should never hardcode gas limits into a contract, and all inputs to CALL that are not just “send all gas” should have gas values provided by the transaction?

Pros

- we don’t have to do anything
- hard coding gas limits has already been considered bad practice for awhile

Cons

- gas is still observable
- existing contracts will continue to be broken with repricings
- relayers must take on more risk, since failing txs won't pay them (referred to as "type-2 breakage" in [7])

–

Interoperability Solutions

Interoperability for meta-transactions is currently not possible at the protocol level17. While it is orthogonal to atomicity, it is still an important aspect of meta-transactions that deserves a proper solution. It is also possible that there are better synergies between some atomicity solutions and interoperability solutions than others. For those reasons, the related proposals have also been included in this discussion.

CallWithSigner 17

Suppose a new opcode is added to the EVM with the same semantics as CALL

, except it takes three additional parameters [v, r, s]

and in the child frame CALLER

returns the resulting ecrecover

of that signature on the hash of the call data.

Pros

- straightforward addition to the EVM
- flexible, composable

Cons

- breaks tx pool validity invariant18
- signatures are not segregated and therefore it's unclear how to compress them with a ZKP19
- couples EVM execution to account authentication

Sponsored Transactions 20

Suppose the current transaction object is amended to support an additional, optional sponsor

signature. Transactions with this signature would first verify the sender's signature + nonce, then it would verify the sponsor's signature and balance to ensure it has at least value + gas_price * gas_limit

.

Pros

- relatively minimal change

Cons

- not flexible
- increased tx pool complexity

Add PAYGASFROM

to Account Abstraction 21

Introduce a new opcode PAYGASFROM whose arguments are gasprice, from, and calldata. Once executed, the from account will be called with the calldata to verify that it plans to pay for the execution of the transaction. Upon successful verification, it will update any replay protection mechanisms and then return control to the original entry point to continue execution.

Please see 10 for more background on account abstraction.

Pros

- provides more utility for account abstraction

Cons

- signatures are not segregated and therefore it's unclear how to compress them with a ZKP[19]
- not applicable to EOAs

Links

Expand to see all

[5]: Oil: adding a second fuel source to the EVM (pre-EIP)

[7]: Meta transactions, Oil, and Karma megathread

[11]: Counter-proposal to oil/karma: per-account gas limits

[12]: Counter-proposal to oil/karma: per-account gas limits

[13]: Counter-proposal to oil/karma: per-account gas limits

[14]: Meta transactions, Oil, and Karma megathread

[15]: Meta transactions, Oil, and Karma megathread

[16]: Counter-proposal to oil/karma: per-account gas limits