This is the first version of a draft for BLS transaction type ERC. The intention of this post is to gather feedback from the wider L2 and Ethereum communities to eventually create a standard for BLS transaction authentication. A single standard makes it much easier for wallets and tooling to support a wide variety of L2s.

Abstract

Introduce a new EIP-2718 transaction type to specify how users can authenticate transactions using BLS signatures.

Motivation

In order for optimistic rollups to remain secure they must make full transaction and signature data available on Ethereum. Although this data can be compressed this is still the dominant cost to rollup users.

One way to reduce this data is to aggregate all the signatures in a rollup batch, and only make the aggregate signature available on Ethereum. Validators and observers could then validate all the transactions in a batch by verifying a single aggregate signature. As an illustration of the potential savings: a typical ETH transfer contains around 120 bytes of which 68 are the signature; if the signature could be removed in favour of aggregate verification this would reduce the data availability costs of that transaction by ~55%. EIP-4844 also provides a path for reducing rollup data availability costs, but this is complementary to BLS signature aggregation and can be worked on by rollup teams in parallel to core developers working on 4844.

Existing transaction types use ECDSA, a signature scheme which doesn't natively support signature aggregation. Optimistic rollups wishing to aggregate signatures need users to authenticate their transactions with a signature scheme that supports aggregation, one such scheme is BLS. In order to see widespread adoption wallets must implement BLS authentication and transaction formatting. Since BLS signing will likely be of use to many optimistic rollups it's also important that the community agrees upon a standard that works for all to avoid ecosystem fragmentation and multiple implementations in wallets.

Specification

Parameters

Constant

Value

MAX_CALLDATA_SIZE

2**24

MAX_ACCESS_LIST_SIZE

2**24

MAX_ACCESS_LIST_STORAGE_KEYS

2**24

BLS_TX_TYPE

Bytes1(0x06)

BLS_MODULUS

52435875175126190479447740508185965837690552500527637822603658699938581184513

Type aliases

Type

Alias

Additional checks

BLSPubKey

Bytes48

x < BLS_MODULUS

BLSSignature

Bytes96

Address

Bytes20

We introduce a new EIP-2718 transaction type, with the format being the single byte BLS_TX_TYPE

followed by an SSZ encoding of the SignedBLSTransaction

container:

class BLSAuth(Container): pubKey: Bytes48 signature: Bytes96

class AccessTuple(Container): address: Address storage_keys: List[Hash, MAX_ACCESS_LIST_STORAGE_KEYS]

class Transaction(Container): chain_id: uint256 nonce: uint64 priority_fee_per_gas: uint256 max_basefee_per_gas: uint256 gas: uint64 to: Union[None, Address] value: uint256 data: ByteList[MAX_CALLDATA_SIZE] access_list: List[AccessTuple, MAX_ACCESS_LIST_SIZE]

class SignedBLSTransaction(Container): message: Transaction auth: BLSAuth

The signature

and pubKey

in BLSAuth are specified by draft-irtf-cfrg-bls-signature-05. The message to be signed is BLS_TX_TYPE || ssz.hash_tree_root(tx)

where tx is an object of type Transaction

container.

Rationale

BLS public key in the transaction

Unlike in ECDSA, the BLS public key cannot be recovered from a message and signature, so it must be provided along with the signature. This may appear to largely defeat the point of BLS signature aggregation, since we would remove an ECDSA signature of 64 bytes and add a BLS public key of 48 bytes. However it's possible for rollups to switch out the BLS key for a much shorter identifier via a mapping stored in state. The identifier could then be re-substituted by validators when verifying the signature.

Unspecified from

address

The from

address of the transaction is not specified. A rollup may choose to maintain a mapping of BLS public key to Ethereum from

address, or they may choose to derive the address from the BLS public key using a deterministic algorithm. How the from

address of this transaction is left out of the scope of this EIP.

Intrinsic gas

The computational cost to verify a BLS signature is much higher that to verify an ECDSA signature. As such implementers of this transaction type will likely need to specify a higher intrinsic gas for this transaction type than other ECDSA authenticated ones. The producer of the aggregated batch will need to calculate two expensive pairing operations to verify each transaction, however optimistic rollup validators will only need to calculate one pairing operation per transaction and one for the whole batch. Different protocols may therefore choose to price intrinsic gas differently, so intrinsic gas specification is left out of the scope of this EIP.

Why BLS12-281?

Although the EVM only currently supports precompiles for the BN254 this transaction signature does not need to be verified within the EVM. Although BN254 is secure now it is on the limit of what is consider acceptable, and as such many projects

eg Zcash, Eth2 use BLS12-381 instead due to its higher bit security. Choosing BLS12-381 for this transaction type makes it more likely to remain compatible with the rest of the Ethereum ecosystem.

Proof of Possession

BLS signature aggregation is susceptible to rogue key attacks. In order to guard against this the verifier must ensure that either all of the messages being verified are unique, or that each of the public keys are accompanied by a proof of possession. A proof of possession is not included in this EIP as it is expected that the verifier will obtain it via a different means.

Some open questions

1. Is it necessary to ssz.hash_tree_root the tx before signing it? Couldn't we just sign the serialised tx? ssz.hash_tree_root can be expensive since it calculates a merkle root.
2. The aim of this EIP is to try to get the ecosystem (wallets and L2s) to agree on a standard for the wire format for a BLS transaction. The EIP doesn't require an consensus changes so I've made it an ERC, however input from Ethereum core devs would be greatly appreciated. The one thing we do want from core devs is an assurance that the transaction type 0x06 won't be re-used for another purpose in Ethereum. The plan is to create a separate Core EIP whose only contents is to reserve the 0x06 transaction type for BLS use, and contains a reference to this ERC.