Signing schemes

As CoW Protocol handles user's orders that were provided off-chain, the authenticity of a user's order needs to be asserted. Authenticity is asserted by the usersigning their order. A user may be an EOA or a smart contract.

Order digest

The order digest is abytes32 that uniquely describes the parameters of an order. It is used as the input to the signing schemes. The digest is computed as described in theEIP-712 .

Replay protection

Replay protection for the signing schemes is achieved by using anEIP-712 domain separator. This domain separator is unique to each deployment of the CoW Protocol settlement contract on a particular chain.

When computing the domain separator, the following parameters are used:

{ name :

"Gnosis Protocol" , version :

"v2" , chainId :

/ *chain ID for the current network: e.g. 1 for mainnet*/ , verifyingContract :

"0x9008D19f58AAbD9eD0D60971565AA8510560ab41" } note CoW Protocol was formerly known as Gnosis Protocol v2. The domain separator is still namedGnosis Protocol as the change would have required a new deployment of the settlement contract. The actual domain separator is the result of hashing the previousEIP712Domain struct with theEIP-712 definedhashStruct function.

tip For convenience, the domain separator is exposed as a public view function in the settlement contract:

- Ethereum mainnet
- Gnosis chain
- Sepolia

Encoding

To encode the order into the order digest, theEIP-712 specification is used:

orderDigest = keccak256("\x19\x01" ‖ domainSeparator ‖ hashStruct(orderStruct)) The components are:

- keccak256
- , the standard unpadded Ethereum hashing function
- "\x19\x01"
- , two bytes
- ‖
- , the byte-concatenation function
- domainSeparator
- , the domain separator as mentioned in the previous section
- hashStruct
- , theidentically named function in the EIP-712 standard

hashStruct pseudocode hashStruct(typeHash, data) = keccak256(typeHash ‖ encodeData(data)) The components are:

- keccak256
- , the standard unpadded Ethereum hashing function
- ‖
- , the byte-concatenation function
- typeHash
- , the hash of the struct type
- encodeData
- , theidentically named function in the EIP-712 standardThe resulting order digest is abytes32 value that is used as the input to the signing schemes.

JavaScript example

Phew, that was a lot of hashing! Feeling dizzy? Don't worry, we got you covered.

The@cowprotocol/contracts package provides a function to compute the order digest for you:

import

{ Order , OrderBalance , OrderKind , domain , hashOrder }

from

"@cowprotocol/contracts"

// Define an order const order : Order =

{ sellToken :

"0x6b175474e89094c44da98b954eedeac495271d0f" ,

// dai buyToken :

"0xdef1ca1fb7fbcdc777520aa7f396b4e015f497ab" ,

// cow sellAmount :

"100000000000000000000000000" ,

// 10m dai buyAmount :

"10000" ,

// 10k cow validTo :

1704067200 , appData :

"0xc85ef7d79691fe79573b1a7064c19c1a9819ebdbd1faaab1a8ec92344438aaf4" ,

// keccak256("cow") feeAmount :

"0" , kind : OrderKind . SELL , partiallyFillable :

false , sellTokenBalance : OrderBalance . ERC20 , buyTokenBalance : OrderBalance . ERC20 , } ;

// hash the order to generate the order digest const digest =

hashOrder (domain (1 ,

"0x9008D19f58AAbD9eD0D60971565AA8510560ab41") , order) ;

// output the digest console . log (Order digest: { digest })

Etherscan Example

For convenience, we also deployed a small helper contract that makes it easy to compute order uids as well as their full encoding for each chain using the Etherscan UI:

- Mainnet
- Gnosis Chain
- Sepolia

Supported schemes

CoW Protocol supports four different signing schemes. These signing schemes are summarized in the following table:

Signing Scheme Gasless EOA Smart Contract eth_sign ✓ ✓ ✗ EIP-712 ✓ ✓ ✗ ERC-1271 ✓ ✗ ✓ PreSign ✗ ✓ ✓ Except for thePreSign scheme, all signing schemes involve signing anorder digest that is based on the message structure ofEIP-712 .

Order book API In the expanded detail sections below,signature refers to the value passed tosignature in theOrder book API .

eth_sign

This signature type is the most commonly supported signing mechanism for EOAs.

The signature is computed as:

signature = ethSign(orderDigest) The components are:

- ethSign
- , using the user's private key to ECDSA-sign a message prefixed with"\x19Ethereum signed message:\n"
- and its length
- orderDigest
- , the order digest

Most Ethereum libraries supportethSign signatures (ethers-js ,web3py).

EIP-712

This signing method, also known as typed structured data signing, is the recommended signing method for EOAs, since the user will be able to see the full order data that is being signed in most wallet implementations.

The signature is computed as:

signature = ecdsaSign(orderDigest) The components are:

- ecdsaSign
- , using the user's private key to ECDSA-sign the message
- orderDigest
- , the order digest

Many Ethereum libraries have some degree of support for signing typed data without building the order digest by hand (for exampleethers-js , andweb3py).

In any case, you may want to read about thedomain separator andencoding the order struct .

ERC-1271

This signing mechanism is the only option that can be used by smart contracts to provide off-chain signatures for orders.

In order to support smart-contract orders, the trading smart contract (the user) must implement theEIP-1271 interface.

signature = eip1271Signature Order book API When submitting anERC-1271 order to the API, thefrom fieldMUST be set to the address of the smart contract that will be signing the order. This is because theERC-1271 signature is not an Ethereum signature, and thus thefrom field cannot be inferred from the signature. The components are:

- eip1271Signature
- , anybytes
- that is a valid signature for the contract for the order. This signature is contract specific.

For an order to be accepted, theeip1271Signature must be valid for theorderDigest message, that is in Solidity:

isValidSignature (orderDigest , eip1271Signature)

== MAGICVALUE

PreSign

This is the only signing method that supports both EOA and smart-contract traders.

Together with submitting aPreSign order, the user must submit the order on-chain. This is done by calling the settlement contractsetPreSignature function:

setPreSignature (orderUid ,

true) ; The components are:

- orderUid
- , the unique identifier of the order

- true
- , whether the order is pre-signed or not (allows for cancelling pre-signed orders)

The signature is an emptybytes string:

signature = 0x Order book API When submitting aPreSign order to the API, thefrom fieldMUST be set to the address of the order owner. This is because thePreSign signature is not an Ethereum signature, and thus thefrom field cannot be inferred from the signature. note If an order was already filled, then pre-signing it doesnot make it tradable again. Edit this page Previous Hooks Next Supported tokens