1

sponsoredCallERC2771

Sponsored transactions with ERC2771 authentication support

If you plan to use ERC-2771 with amulticall method or any other method usingdelegateCall() Please read carefully the sectionAvoid ERC-2771-risks If you are using@gelatonetwork/relay-sdk v3 or contracts from the package@gelatonetwork/relay-context v2 please follow thismigration guide to migrate to the new versions. After reading this page:

- You'll know how to use thesponsoredCallERC2771
- SDK method. This will give your user's a gasless UX requiring a user signature. This uses the1Balance
- payment method, allowing you to sponsor some/all of your user's gas costs.
- You'll learn about how toincorporateERC2771Context
- into your contract for_msgSender()
- support.
- You'll see some code which will help you send a relay request within minutes. *

Overview

ThesponsoredCallERC2771 method uses both a sponsor API key and a user's signature, like that from MetaMask, to securely sponsor gasless transactions. Payments are made via theGelato 1Balance method.

Gelato Relay SDK has various methods for handling sponsored ERC2771 transactions. The most straightforward issponsoredCallERC2771 , which handles both signing and sending in one step. If you need to separate these processes, other SDK methods are available.

SDK Methods

sponsoredCallERC2771

This method initiates the signing of ERC2771 requests with the providedBrowserProvider orWallet . Once the signature is obtained, the request is forwarded to Gelato.

```

Copy constsponsoredCallERC2771=async( request:CallWithERC2771Request|CallWithConcurrentERC2771Request, signerOrProvider:ethers.BrowserProvider|ethers.Signer, sponsorApiKey:string, options?:RelayRequestOptions ):Promise

```

Arguments

- request
- : Thebody
- of the request intended for sending.
- signerOrProvider
- : a valid provider connected to RPC or a signer.
- sponsorApiKey
- : an API key used to authenticate your sponsorship.
- options
- : an object for specifyingoptional parameters
- .
-

Response

```

Copy typeRelayResponse={ taskId:string; };

```

- taskId
- : a unique task ID which can be used fortracking your request
- .
-

getSignatureDataERC2771

This method starts the signing process for ERC2771 requests using the givenBrowserProvider orSigner . After capturing the signature, it returns both the signature and the message. This collected data can then be used with thesponsoredCallERC2771WithSignature method to send the request to Gelato.

```

Copy getSignatureDataERC2771=( request:CallWithERC2771Request|CallWithConcurrentERC2771Request, signerOrProvider:ethers.BrowserProvider|ethers.Signer, type:ERC2771Type ):Promise

```

Arguments

- request
- : this is therequest body
- used to send a request.
- signerOrProvider
- : a valid provider connected to RPC or a signer.
- type
- :SponsoredCall
- for asequential
- flow orConcurrentSponsoredCall
- for aconcurrent
- flow.
-

Response

```

Copy typeSignatureData=ConcurrentSignatureData |SequentialSignatureData;

typeConcurrentSignatureData={ struct:CallWithConcurrentERC2771Struct; signature:string; };

typeSequentialSignatureData={ struct:CallWithERC2771Struct; signature:string; };

```

- struct
- : EIP-712 message data.
- signature
- : EIP-712 signature.
-

getDataToSignERC2771

This method provides the message data intended for external signing along with the EIP-712 typed data. After obtaining the signature, the request can be dispatched using thesponsoredCallERC2771WithSignature method.

```

Copy getDataToSignERC2771=( request:CallWithERC2771Request|CallWithConcurrentERC2771Request, type:ERC2771Type, signerOrProvider?:ethers.BrowserProvider|ethers.Signer, ):Promise

```

Arguments

- request
- : Thebody
- of the request intended for sending.
- type
- :SponsoredCall
- for asequential
- flow orConcurrentSponsoredCall
- for aconcurrent
- flow.
- signerOrProvider
- (optional): A provider needed in a sequential flow to obtain the nonce from the smart contract. If you're providing the

nonce within your request or if you're using the concurrent flow, this parameter isn't necessary.

Response

```

Copy typePayloadToSign=ConcurrentPayloadToSign|SequentialPayloadToSign;

typeConcurrentPayloadToSign={ struct:CallWithConcurrentERC2771Struct;
typedData:SponsoredCallConcurrentERC2771PayloadToSign };

typeSequentialPayloadToSign={ struct:CallWithERC2771Struct; typedData:SponsoredCallERC2771PayloadToSign };

```

- struct
- : EIP-712 message data.
- typedData
- : EIP-712 typed data.

sponsoredCallERC2771WithSignature

This method sends pre-signed requests to Gelato.

```

Copy sponsoredCallERC2771WithSignature=async( struct:SignatureData["struct"], signature:SignatureData["signature"],
sponsorApiKey:string, options?:RelayRequestOptions ):Promise

```

Arguments

- struct
- : EIP-712 message data returned from the signing methods.
- signature
- : EIP-712 signature returned after signing the request.
- sponsorApiKey
- : an API key used to authenticate your sponsorship.
- options
- : an object for specifyingoptional parameters
- .

Response

```

Copy typeRelayResponse={ taskId:string; };

```

- taskId
- : a unique task ID which can be used fortracking your request
- .

Optional Parameters

SeeOptional Parameters .

Sending a Request

As of today, we support two distinct ways of sendingsponsoredCallERC2771 requests:

1. Sequentially
2. : This approach ensures that each request is ordered and validated against thenonce
3. stored on-chain. You have two options in this method:
4.

    - Fetch the currentnonce
5.
    - value from the smart contract yourself and include it with your request.
6.
    - Allow the relay-sdk to fetch thenonce
7.
    - value for you when handling your relay request.
8. *
9. Concurrently
10. : This method enables you to send multiple transactions simultaneously. Replay protection is achieved using a hash-basedsalt
11. mechanism. Again, you have two options:
12.
    - Provide your ownsalt
13.
    - value.
14.
    - Allow the relay-sdk to generate a uniquesalt
15.
    - value for you when processing your relay request.
16. *
17.

By defaultsponsoredCallERC2771 requests are using the sequential method.

Concurrent ERC2771 support has been introduced in the relay-sdk version5.1.0 . Please make sure that your package is up-to-date to start using it.

Request Body

```

Copy typeSequentialERC2771Request={ chainId:BigNumberish; target:string; data:BytesLike; user:string; userDeadline?:BigNumberish; isConcurrent?:false; userNonce?:BigNumberish; }; typeConcurrentERC2771Request={ chainId:BigNumberish; target:string; data:BytesLike; user:string; userDeadline?:BigNumberish; isConcurrent:true; userSalt?:string };

```

Common Parameters:

- chainId
- : the chain ID of the chain where thetarget
- smart contract is deployed.
- target
- : the address of the target smart contract.
- data
- : encoded payload data (usually a function selector plus the required arguments) used to call the requiredtarget
- address.
- user
- : the address of the user's EOA.
- userDeadline
- :optional,
- the amount of time in seconds that a user is willing for the relay call to be active in the relay backend before it is dismissed.
-
    - This way the user knows that if the transaction is not sent within a certain timeframe, it will expire. Without this, an adversary could pick up the transaction in the mempool and send it later. This could transfer money, or change state at a point in time which would be highly undesirable to the user.
- *
-

Parameters For Sequential Requests:

- isConcurrent
- :false
- (default)
- ,optional,
- represents that the users' requests are validated based on a nonce, which enforces them to be processed sequentially.

- userNonce
- :optional
- , this nonce, akin to Ethereum nonces, is stored in a local mapping on the relay contracts. It serves to enforce the nonce ordering of relay calls if the user requires sequential processing. If this parameter is omitted, the relay-sdk will automatically query the current value on-chain.
-

Parameters For Concurrent Requests:

- isConcurrent
- :true
- , indicates that the users' requests are validated based on a unique salt, allowing them to be processed concurrently. Replay protection is still ensured by permitting each salt value to be used only once.
- userSalt
- :optional
- , this is a bytes32 hash that is used for replay protection. If the salt is not provided then relay-sdk would generate a unique value based on a random seed and a timestamp.
-

Example Code

For your testing, Gelato has deployed a simple contract which implements logic to increment a counter withERC2771 support.

- CounterERC2771.sol
- : deployed at the address0x00172f67db60E5fA346e599cdE675f0ca213b47b
- onthese networks
- .
-

CounterERC2771.sol 's counter is special because it implementsERC-2771 _msgSender authentication to allow for secure whitelisting based on the identity of the original off-chain relay request originator, which has been verified using a user signature.

Furthermore, to set your trusted forwarder, you need the address forGelatoRelay1BalanceERC2771.sol that you can findhere .

1. Deploy anERC2771Context compatible contract

```

Copy // SPDX-License-Identifier: MIT pragmasolidity0.8.17;

import{ ERC2771Context }from"@gelatonetwork/relay-context/contracts/vendor/ERC2771Context.sol";

// Importing ERC2771Context gives access to: // 1. An immutable trusted forwarder address // 2. function isTrustedForwarder // to verify an input address matches the trustedForwarder address // 3. function _msgSender() // which decodes the user's address from the calldata // _msgSender() can now be used to refer to user safely // instead of msg.sender (which is Gelato Relay in this case). // 4. function _msgData() // which decodes the function signature from the calldata contractCounterERC2771isERC2771Context{ // Here we have a mapping that maps a counter to an address mapping(address=>uint256)publiccontextCounter;

eventIncrementCounter(address_msgSender);

// ERC2771Context: setting the immutable trustedForwarder variable constructor(addresstrustedForwarder)ERC2771Context(trustedForwarder) {}

// incrementContext is the target function to call // This function increments a counter variable which is // mapped to every _msgSender(), the address of the user. // This way each user off-chain has their own counter // variable on-chain. functionincrement()external{ // Remember that with the context shift of relaying, // where we would use msg.sender before, // this now refers to Gelato Relay's address, // and to find the address of the user, // which has been verified using a signature, // please use _msgSender()!

// If this contract was not not called by the // trusted forwarder, _msgSender() will simply return // the value of msg.sender instead.

// Incrementing the counter mapped to the _msgSender! contextCounter[_msgSender()]++;

// Emitting an event for testing purposes emitIncrementCounter(_msgSender()); } }

```

1. Import GelatoRelaySDK into your front-end .js project

```

Copy import{ GelatoRelay,SponsoredCallERC2771Request }from"@gelatonetwork/relay-sdk"; constrelay=newGelatoRelay();

```

1. Send the payload to Gelato

This is an example using Gelato'sCounterERC2771.sol which is deployed orthese networks .

```

Copy // Set up on-chain variables, such as target address constcounter="0x00172f67db60E5fA346e599cdE675f0ca213b47b"; constabi=["function increment()"]; constprovider=newethers.BrowserProvider(window.ethereum); constsigner=awaitprovider.getSigner(); constuser=awaitsigner.getAddress();

// Generate the target payload constcontract=newethers.Contract(counter,abi,signer); const{data}=awaitcontract.incrementContext.populateTransaction();

// Populate a relay request constrequest:CallWithERC2771Request={ chainId:(awaitprovider.getNetwork()).chainId, target:counter; data: data; user: user; };

// Without a specific API key, the relay request will fail! // Go to https://relay.gelato.network to get a testnet API key with 1Balance. // Send a relay request using Gelato Relay! constrelayResponse=awaitrelay.sponsoredCallERC2771(request,provider,apiKey);

```

Previous ERC-2771 (recommended) Next callWithSyncFeeERC2771 Last updated2 days ago On this page *Overview * SDK Methods * sponsoredCallERC2771 * getSignatureDataERC2771 * getDataToSignERC2771 * sponsoredCallERC2771WithSignature * Optional Parameters * Sending a Request * Example Code * 1. Deploy an ERC2771Context compatible contract