Signing and Sending Transactions

Transactions in Sui represent calls to specific functionality (like calling a smart contract function) that execute on inputs to define the result of the transaction.

Inputs can either be an object reference (either to an owned object, an immutable object, or a shared object), or an encoded value (for example, a vector of bytes used as an argument to a Move call). After a transaction is constructed, usually through using programmable transaction blocks (PTBs), the user signs the transaction and submits it to be executed on chain.

The signature is provided with the private key owned by the wallet, and its public key must be consistent with the transaction sender's Sui address.

Sui uses a SuiKeyPair to produce the signature, which commits to the Blake2b hash digest of the intent message (intent || bcs bytes of tx_data). The signature schemes currently supported are Ed25519 Pure , ECDSA Secp256k1 , ECDSA Secp256r1 , Multisig , and zkLogin .

You can instantiate Ed25519 Pure , ECDSA Secp256k1 , and ECDSA Secp256r1 using SuiKeyPair and use it to sign transactions. Note that this guide does not apply to Multisig and zkLogin , please refer to their own pages (Multisig and zkLogin respectively) for instructions.

With a signature and the transaction bytes, a transaction can be submitted to be executed.

The following high-level process describes the overall workflow for constructing, signing and executing an on-chain transaction:

If you want to use a specific gas coin, first find the gas coin object ID to be used to pay for gas, and explicitly use that in the PTB. If there is no gas coin object, use the splitCoin transaction to create a gas coin object. The split coin transaction should be the first transaction call in the PTB.

The following examples demonstrate how to sign and execute transactions using Rust, TypeScript, or the Sui CLI.

There are various ways to instantiate a key pair and to derive its public key and Sui address using the Sui TypeScript SDK.

The full code example below can be found under crates/sui-sdk .

There are various ways to instantiate a SuiKeyPair and to derive its public key and Sui address using the Sui Rust SDK.

Next, sign transaction data constructed using an example programmable transaction block with default gas coin, gas budget, and gas price. See Building Programmable Transaction Blocks for more information.

Commit a signature to the Blake2b hash digest of the intent message (intent || bcs bytes of tx_data).

Finally, submit the transaction with the signature.

When using the Sui CLI for the first time, it creates a local file in ~/.sui/keystore on your machine with a list of private keys (encoded as Base64 encoded flag || 32-byte-private-key). You can use any key to sign transactions by specifying its address. Use sui keytool list to see a list of addresses.

There are three ways to initialize a key:

Create a transfer transaction in the CLI. Set the $SUI_ADDRESS to the one corresponding to the keypair used to sign. $GAS_COIN_ID refers to the object ID that is owned by the sender to be used as gas. $GAS_BUDGET refers to the budget used to execute transaction. Then sign with the private key corresponding to the sender address. $MNEMONICS refers to 12/15/18/21/24 words from the wordlist, e.g. "retire skin goose will hurry this field stadium drastic label husband venture cruel toe wire". Refer to Keys and Addresses for more.

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.

Workflow

The following high-level process describes the overall workflow for constructing, signing and executing an on-chain transaction:

If you want to use a specific gas coin, first find the gas coin object ID to be used to pay for gas, and explicitly use that in the PTB. If there is no gas coin object, use the splitCoin transaction to create a gas coin object. The split coin transaction should be the first

transaction call in the PTB.

The following examples demonstrate how to sign and execute transactions using Rust, TypeScript, or the Sui CLI.

There are various ways to instantiate a key pair and to derive its public key and Sui address using the Sui TypeScript SDK.

The full code example below can be found under crates/sui-sdk .

There are various ways to instantiate a SuiKeyPair and to derive its public key and Sui address using the Sui Rust SDK.

Next, sign transaction data constructed using an example programmable transaction block with default gas coin, gas budget, and gas price. See Building Programmable Transaction Blocks for more information.

Commit a signature to the Blake2b hash digest of the intent message (intent || bcs bytes of tx_data).

Finally, submit the transaction with the signature.

When using the Sui CLI for the first time, it creates a local file in ~/.sui/keystore on your machine with a list of private keys (encoded as Base64 encoded flag || 32-byte-private-key). You can use any key to sign transactions by specifying its address. Use sui keytool list to see a list of addresses.

There are three ways to initialize a key:

Create a transfer transaction in the CLI. Set the $SUI_ADDRESS to the one corresponding to the keypair used to sign. $GAS_COIN_ID refers to the object ID that is owned by the sender to be used as gas. $GAS_BUDGET refers to the budget used to execute transaction. Then sign with the private key corresponding to the sender address. $MNEMONICS refers to 12/15/18/21/24 words from the wordlist, e.g. "retire skin goose will hurry this field stadium drastic label husband venture cruel toe wire". Refer to Keys and Addresses for more.

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.

Examples

The following examples demonstrate how to sign and execute transactions using Rust, TypeScript, or the Sui CLI.

There are various ways to instantiate a key pair and to derive its public key and Sui address using the Sui TypeScript SDK.

The full code example below can be found under crates/sui-sdk .

There are various ways to instantiate a SuiKeyPair and to derive its public key and Sui address using the Sui Rust SDK.

Next, sign transaction data constructed using an example programmable transaction block with default gas coin, gas budget, and gas price. See Building Programmable Transaction Blocks for more information.

Commit a signature to the Blake2b hash digest of the intent message (intent || bcs bytes of tx_data).

Finally, submit the transaction with the signature.

When using the Sui CLI for the first time, it creates a local file in ~/.sui/keystore on your machine with a list of private keys (encoded as Base64 encoded flag || 32-byte-private-key). You can use any key to sign transactions by specifying its address. Use sui keytool list to see a list of addresses.

There are three ways to initialize a key:

Create a transfer transaction in the CLI. Set the $SUI_ADDRESS to the one corresponding to the keypair used to sign. $GAS_COIN_ID refers to the object ID that is owned by the sender to be used as gas. $GAS_BUDGET refers to the budget used to execute transaction. Then sign with the private key corresponding to the sender address. $MNEMONICS refers to 12/15/18/21/24 words from the wordlist, e.g. "retire skin goose will hurry this field stadium drastic label husband venture cruel toe wire". Refer to Keys and Addresses for more.

Beginning with the Sui v1.24.1 release , the --gas-budget option is no longer required for CLI commands.