Hasing with keccak256 • Stylus by Example

Hashing with keccak256

Keccak256 is a cryptographic hash function that takes an input of an arbitrary length and produces a fixed-length output of 256 bits.

Keccak256 is a member of theSHA-3 family of hash functions.

keccak256 computes the Keccak-256 hash of the input.

Some use cases are:

- Creating a deterministic unique ID from a input
- Commit-Reveal scheme
- Compact cryptographic signature (by signing the hash instead of a larger input)

Here we will usestylus-sdk::crypto::keccak to calculate the keccak256 hash of the input data:

note This code has yet to be audited. Please use at your own risk. pub

fn

keccak < T :

AsRef < [u8]

(bytes :

T)

->

B256

Full Example code:

src/main.rs

// Only run this as a WASM if the export-abi feature is not set.

![cfg_attr(not(any(feature =

"export-abi" , test)), no_main)] extern

crate

alloc ;

/// Import items from the SDK. The prelude contains common traits and macros. use

stylus_sdk :: { alloy_primitives :: { U256 ,

Address ,

FixedBytes } ,

abi :: Bytes ,

prelude :: * ,

crypto :: keccak } ; use

alloc :: string :: String ; use

alloc :: vec :: Vec ; // Becauce the naming of alloy_primitives and alloy_sol_types is the same, so we need to re-name the types in alloy_sol_types use alloy_sol_types :: { sol_data :: { Address

as

SOLAddress ,

String

as

SOLString ,

Bytes

as

SOLBytes ,

* } ,

SolType } ; use

alloy_sol_types :: sol ;

// Define error sol!

{ error DecodedFailed ( ) ; }

// Error types for the MultiSig contract

[derive(SolidityError)]

pub

enum

HasherError { DecodedFailed (DecodedFailed) }

[solidity_storage]

[entrypoint]

pub

struct

Hasher

{ } /// Declare that Hasher is a contract with the following external methods.

[public]

impl

Hasher

{

// Encode the data and hash it pub

fn

encode_and_hash (& self , target :

Address , value :

U256 , func :

```
String , data :

Bytes , timestamp :

U256 )

->

FixedBytes < 32

{ // define sol types tuple type

TxIdHashType

=

( SOLAddress ,

Uint < 256

      ,

SOLString ,

SOLBytes ,

Uint < 256

      ) ; // set the tuple let tx_hash_data =

( target , value , func , data , timestamp ) ; // encode the tuple let tx_hash_data_encode =

TxIdHashType :: abi_encode_sequence ( & tx_hash_data ) ; // hash the encoded data keccak ( tx_hash_data_encode ) .
into ( ) }

// This should always return true pub

fn

encode_and_decode ( & self , address :

Address , amount :

U256 )

->

Result < bool ,

HasherError

{ // define sol types tuple type

TxIdHashType

=

( SOLAddress ,

Uint < 256

      ) ; // set the tuple let tx_hash_data =

( address , amount ) ; // encode the tuple let tx_hash_data_encode =

TxIdHashType :: abi_encode_sequence ( & tx_hash_data ) ;

let validate =

true ;

// Check the result match

TxIdHashType :: abi_decode_sequence ( & tx_hash_data_encode , validate )
```

```
{ Ok ( res )

=>

Ok ( res == tx_hash_data ) , Err ( _ )

=>

{ return

Err ( HasherError :: DecodedFailed ( DecodedFailed { } ) ) ; } , } }

// Packed encode the data and hash it, the same result with the following one pub

fn

packed_encode_and_hash_1 ( & self , target :

Address , value :

U256 , func :

String , data :

Bytes , timestamp :

U256 ) ->

FixedBytes < 32

{ // define sol types tuple type

TxIdHashType

=

( SOLAddress ,

Uint < 256

    ,

SOLString ,

SOLBytes ,

Uint < 256

    ) ; // set the tuple let tx_hash_data =

( target , value , func , data , timestamp ) ; // encode the tuple let tx_hash_data_encode_packed =

TxIdHashType :: abi_encode_packed ( & tx_hash_data ) ; // hash the encoded data keccak ( tx_hash_data_encode_packed ) . into ( ) }

// Packed encode the data and hash it, the same result with the above one pub

fn

packed_encode_and_hash_2 ( & self , target :

Address , value :

U256 , func :

String , data :

Bytes , timestamp :

U256 ) ->

FixedBytes < 32

{ // set the data to arrary and concat it directly let tx_hash_data_encode_packed =
[ & target . to_vec ( ) ,
& value . to_be_bytes_vec ( ) , func . as_bytes ( ) ,
& data . to_vec ( ) ,
& timestamp . to_be_bytes_vec ( ) ] . concat ( ) ; // hash the encoded data keccak ( tx_hash_data_encode_packed ) . into ( ) }
// The func example: "transfer(address,uint256)" pub
fn
encode_with_signature ( & self , func :
String , address :
Address , amount :
U256 )
->
Vec < u8
{ type
TransferType
=
( SOLAddress ,
Uint < 256
     ) ; let tx_data =
( address , amount ) ; let data =
TransferType :: abi_encode_sequence ( & tx_data ) ; // Get function selector let hashed_function_selector :
FixedBytes < 32
=
keccak ( func . as_bytes ( ) . to_vec ( ) ) . into ( ) ; // Combine function selector and input data (use abi_packed way) let calldata =
[ & hashed_function_selector [ .. 4 ] ,
& data ] . concat ( ) ; calldata }
// The func example: "transfer(address,uint256)" pub
fn
encode_with_signature_and_hash ( & self , func :
String , address :
Address , amount :
U256 )
->
FixedBytes < 32
{ type
TransferType
=
( SOLAddress ,
Uint < 256
        ) ; let tx_data =
( address , amount ) ; let data =
TransferType :: abi_encode_sequence ( & tx_data ) ; // Get function selector let hashed_function_selector :
FixedBytes < 32
=
keccak ( func . as_bytes ( ) . to_vec ( ) ) . into ( ) ; // Combine function selector and input data (use abi_packed way) let calldata =
[ & hashed_function_selector [ .. 4 ] ,
& data ] . concat ( ) ; keccak ( calldata ) . into ( ) } }
```

Cargo.toml

```
[ package ] name
=
"stylus-encode-hashing" version
=
"0.1.0" edition
=
"2021"
[ dependencies ] alloy-primitives
=
"=0.7.6" alloy-sol-types
=
"=0.7.6" mini-alloc
=
"0.4.2" stylus-sdk
=
"0.6.0" hex
=
"0.4.3" sha3
=
"0.10.8"
[ features ] export-abi
=
[ "stylus-sdk/export-abi" ] debug
=
[ "stylus-sdk/debug" ]
[ lib ] crate-type
```

```
=

[ "lib" ,

"cdylib" ]

[ profile.release ] codegen-units

=

1 strip

=

true lto

=

true panic

=

"abort" opt-level

=

"s"
```

Edit this page Previous Abi Decode Next Bytes In Bytes Out