Intent Signing

In Sui, an intent is a compact struct that serves as the domain separator for a message that a signature commits to. The data that the signature commits to is an intent message. All signatures in Sui must commit to an intent message, instead of the message itself.

In previous releases, Sui used a special Signable trait that attached the Rust struct name as a prefix to the serialized data. This is not ideal because it's:

The intent signing standard provides a compact domain separator to the data being signed for both user signatures and authority signatures. It has several benefits, including:

The IntentMessage struct consists of the intent and the serialized data value.

To create an intent struct, include the IntentScope (what the type of the message is), IntentVersion (what version the network supports), and AppId (what application that the signature refers to).

To see a detailed definition for each field, see each enum definition in the source code .

The serialization of an Intent is a 3-byte array where each field is represented by a byte.

The serialization of an IntentMessage is the 3 bytes of the intent concatenated with the BCS serialized message.

To create a user signature, construct an intent message first, and create the signature over the 32-byte Blake2b hash of the BCS serialized value of the intent message of the transaction data (intent || message).

Here is an example in Rust:

Here is an example in TypeScript:

Under the hood, the new_secure method in Rust and the signData method in TypesScript does the following:

The authority signature is created using the protocol key. The data that it commits to is also an intent message intent || message . See all available intent scopes in the source code

When an authority request to join the network, the protocol public key and its proof of possession (PoP) are required to be submitted. PoP is required to prevent rogue key attack .

The proof of possession is a BLS signature created using the authority's protocol private key, committed over the following message: intent || pubkey || address || epoch . Here intent is serialized to [5, 0, 0] representing an intent with scope as "Proof of Possession", version as "V0" and app_id as "Sui". pubkey is the serialized public key bytes of the authority's BLS protocol key. address is the account address associated with the authority's account key. epoch is serialized to [0, 0, 0, 0, 0, 0, 0, 0] .

To generate a proof of possession in Rust, see implementation at fn generate_proof_of_possession . For test vectors, see fn test_proof_of_possession .

Motivation

In previous releases, Sui used a special Signable trait that attached the Rust struct name as a prefix to the serialized data. This is not ideal because it's:

The intent signing standard provides a compact domain separator to the data being signed for both user signatures and authority signatures. It has several benefits, including:

The IntentMessage struct consists of the intent and the serialized data value.

To create an intent struct, include the IntentScope (what the type of the message is), IntentVersion (what version the network supports), and AppId (what application that the signature refers to).

To see a detailed definition for each field, see each enum definition in the source code .

The serialization of an Intent is a 3-byte array where each field is represented by a byte.

The serialization of an IntentMessage is the 3 bytes of the intent concatenated with the BCS serialized message.

To create a user signature, construct an intent message first, and create the signature over the 32-byte Blake2b hash of the BCS serialized value of the intent message of the transaction data (intent || message).

Here is an example in Rust:

Here is an example in TypeScript:

Under the hood, the new_secure method in Rust and the signData method in TypesScript does the following:

The authority signature is created using the protocol key. The data that it commits to is also an intent message intent || message . See all available intent scopes in the source code

When an authority request to join the network, the protocol public key and its proof of possession (PoP) are required to be submitted. PoP is required to prevent rogue key attack .

The proof of possession is a BLS signature created using the authority's protocol private key, committed over the following message: intent || pubkey || address || epoch . Here intent is serialized to [5, 0, 0] representing an intent with scope as "Proof of Possession", version as "V0" and app_id as "Sui". pubkey is the serialized public key bytes of the authority's BLS protocol key. address is the account address associated with the authority's account key. epoch is serialized to [0, 0, 0, 0, 0, 0, 0, 0] .

To generate a proof of possession in Rust, see implementation at fn generate_proof_of_possession . For test vectors, see fn test_proof_of_possession .

Structs

The IntentMessage struct consists of the intent and the serialized data value.

To create an intent struct, include the IntentScope (what the type of the message is), IntentVersion (what version the network supports), and AppId (what application that the signature refers to).

To see a detailed definition for each field, see each enum definition in the source code .

The serialization of an Intent is a 3-byte array where each field is represented by a byte.

The serialization of an IntentMessage is the 3 bytes of the intent concatenated with the BCS serialized message.

To create a user signature, construct an intent message first, and create the signature over the 32-byte Blake2b hash of the BCS serialized value of the intent message of the transaction data (intent || message).

Here is an example in Rust:

Here is an example in TypeScript:

Under the hood, the new_secure method in Rust and the signData method in TypesScript does the following:

The authority signature is created using the protocol key. The data that it commits to is also an intent message intent || message . See all available intent scopes in the source code

When an authority request to join the network, the protocol public key and its proof of possession (PoP) are required to be submitted. PoP is required to prevent rogue key attack .

The proof of possession is a BLS signature created using the authority's protocol private key, committed over the following message: intent || pubkey || address || epoch . Here intent is serialized to [5, 0, 0] representing an intent with scope as "Proof of Possession", version as "V0" and app_id as "Sui". pubkey is the serialized public key bytes of the authority's BLS protocol key. address is the account address associated with the authority's account key. epoch is serialized to [0, 0, 0, 0, 0, 0, 0, 0] .

To generate a proof of possession in Rust, see implementation at fn generate_proof_of_possession . For test vectors, see fn test_proof_of_possession .

User Signature

To create a user signature, construct an intent message first, and create the signature over the 32-byte Blake2b hash of the BCS serialized value of the intent message of the transaction data (intent || message).

Here is an example in Rust:

Here is an example in TypeScript:

Under the hood, the new_secure method in Rust and the signData method in TypesScript does the following:

The authority signature is created using the protocol key. The data that it commits to is also an intent message intent || message . See all available intent scopes [in the source code]

When an authority request to join the network, the protocol public key and its proof of possession (PoP) are required to be submitted. PoP is required to prevent [rogue key attack] .

The proof of possession is a BLS signature created using the authority's protocol private key, committed over the following message: intent || pubkey || address || epoch . Here intent is serialized to [5, 0, 0] representing an intent with scope as "Proof of Possession", version as "V0" and app_id as "Sui". pubkey is the serialized public key bytes of the authority's BLS protocol key. address is the account address associated with the authority's account key. epoch is serialized to [0, 0, 0, 0, 0, 0, 0, 0] .

To generate a proof of possession in Rust, see implementation at fn generate_proof_of_possession . For test vectors, see fn test_proof_of_possession .

Authority Signature

The authority signature is created using the protocol key. The data that it commits to is also an intent message intent || message . See all available intent scopes [in the source code]

When an authority request to join the network, the protocol public key and its proof of possession (PoP) are required to be submitted. PoP is required to prevent [rogue key attack] .

The proof of possession is a BLS signature created using the authority's protocol private key, committed over the following message: intent || pubkey || address || epoch . Here intent is serialized to [5, 0, 0] representing an intent with scope as "Proof of Possession", version as "V0" and app_id as "Sui". pubkey is the serialized public key bytes of the authority's BLS protocol key. address is the account address associated with the authority's account key. epoch is serialized to [0, 0, 0, 0, 0, 0, 0, 0] .

To generate a proof of possession in Rust, see implementation at fn generate_proof_of_possession . For test vectors, see fn test_proof_of_possession .