Contract State Encryption

Encryption at deployment of contract

When a contract is executed on chain the state of the contracts needs to be encrypted so that observers can not see the computation that is initialized. The contract should be able to call certain functions inside the enclave and store the contract state on-chain.

A contract can call 3 different functions:write_db(field_name, value) ,read_db(field_name) , andremove_db(field_name) . It is important that thefield_name remains constant between contract calls.

We will go over the different steps associated with the encryption of the contract state.

1. Createcontract_key

Thecontract_key is the encryption key for the contract state and is a combination of two values:signer_id || authenticated_contract_key . Every contract has its own unforgeable encryption key. The concatenation of the values is what makes every unique and this is important for several reasons

1. Make sure the state of two contracts with the same code is different
2. Make sure a malicious node runner won't be able to locally encrypt transactions with it's own encryption key, and then decrypt the resulting state with the fake key
3.

so to reiterate, every contract on Secret Network has its own unique and unforgeable encryption keycontract_key

This process of creatingcontract_key is started when the Secret contract is deployed on-chain. Firstauthentication_key is generated using HDKF-SHA256 inside the enclave from the following values:

- consensus_state_ikm
- HDK-salt
- signer_id
-

```

Copy signer_id = sha256(concat(msg_sender, block_height));

authentication_key = hkdf({ salt: hkdf_salt, info: "contract_key", ikm: concat(consensus_state_ikm, signer_id), });

```

From theauthentication_key createauthenticated_contract_key by calling the hmac-SHA256 hash function with the contractcode_hash as hashing data.

This step makes sure the key is unique for every contracts with different code.

```

Copy authenticated_contract_key = hmac_sha256({ key: authentication_key, data: code_hash, });

```

Lastly concat thesigner_id andauthenticated_contract_key to createcontract_key . This step makes it so the key is unforgeable as the key can only be recreated with the currentsigner_id

```

Copy contract_key = concat(signer_id, authenticated_contract_key);

```

1. At execution sharecontract_key with enclave

Every time a contract execution is called,contract_key should be sent to the enclave. In the enclave, the following verification needs to happen to proof a genuinecontract_key

```

Copy signer_id = contract_key.slice(0, 32); expected_contract_key = contract_key.slice(32, 64);
```

authentication_key = hkdf({ salt: hkdf_salt, info: "contract_key", ikm: concat(consensus_state_ikm, signer_id), });

calculated_contract_key = hmac_sha256({ key: authentication_key, data: code_hash, });

assert(calculated_contract_key == expected_contract_key);

```

1. Callback function logic

write_db(field_name, value)

```

Copy encryption_key = hkdf({ salt: hkdf_salt, ikm: concat(consensus_state_ikm, field_name, contract_key), });

encrypted_field_name = aes_128_siv_encrypt({ key: encryption_key, data: field_name, });

current_state_ciphertext = internal_read_db(encrypted_field_name);

if (current_state_ciphertext == null) { // field_name doesn't yet initialized in state ad = sha256(encrypted_field_name); } else { // read previous_ad, verify it, calculate new ad previous_ad = current_state_ciphertext.slice(0, 32); // first 32 bytes/256 bits current_state_ciphertext = current_state_ciphertext.slice(32); // skip first 32 bytes

aes_128_siv_decrypt({ key: encryption_key, data: current_state_ciphertext, ad: previous_ad, }); // just to authenticate previous_ad ad = sha256(previous_ad); }

new_state_ciphertext = aes_128_siv_encrypt({ key: encryption_key, data: value, ad: ad, });

new_state = concat(ad, new_state_ciphertext);

internal_write_db(encrypted_field_name, new_state);

```

read_db(field_name)

```

Copy encryption_key=hkdf({ salt:hkdf_salt, ikm:concat(consensus_state_ikm,field_name,contract_key), });

encrypted_field_name=aes_128_siv_encrypt({ key:encryption_key, data:field_name, });

current_state_ciphertext=internal_read_db(encrypted_field_name);

if(current_state_ciphertext==null) { // field_name doesn't yet initialized in state returnnull; }

// read ad, verify it ad=current_state_ciphertext.slice(0,32);// first 32 bytes/256 bits
current_state_ciphertext=current_state_ciphertext.slice(32);// skip first 32 bytes
current_state_plaintext=aes_128_siv_decrypt({ key:encryption_key, data:current_state_ciphertext, ad:ad, });

returncurrent_state_plaintext;

```

remove_db(field_name)

Very similar toread_db .

```

Copy encryption_key=hkdf({ salt:hkdf_salt, ikm:concat(consensus_state_ikm,field_name,contract_key), });

encrypted_field_name=aes_128_siv_encrypt({ key:encryption_key, data:field_name, });

internal_remove_db(encrypted_field_name);

``` Last updated11 months ago On this page *Encryption at deployment of contract * 1. Create contract_key * 2. At execution share contract_key with enclave * 3. Callback function logic

Was this helpful? Edit on GitHub Export as PDF
```