Access Keys

In all blockchains, users control their accounts by holding aprivate key (a secret only they know) and using it to signtransactions .

NEAR accounts present theunique feature of being able to hold multipleAccess Keys , each with itsown set of permissions . We distinguish two types of Keys:

1. Full-Access Keys
2. : Have full control over the account, and shouldnever be shared
3. Function-Call Keys
4. : Can sign calls to specific contract, and aremeant to be shared

Full-Access Keys

As the name suggests,Full-Access keys have full control of an account, meaning they can be used to signtransactions doing any action in your account's behalf:

1. Transfer NEAR Ⓝ
2. Delete your account or create sub-accounts of it
3. Add or remove Access Keys
4. Deploy a smart contract in the account
5. Call methods on any contract

You should never share yourFull-Access , otherwise you are givingtotal control over the account .

tip Thefirst Full-Access Key of an account is added when the account iscreated

Function-Call Keys

Function-Call keys can only sign transactions calling aspecific contract , and donot allow toattach NEAR tokens to the call. They are defined by three attributes:

1. receiver_id
2. : Thecontract
3. which the key allows to call. No other contract can be called using this key
4. method_names
5. (Optional): The contract'smethods
6. the key allows to call. If omitted, all methods can be called
7. allowance
8. (Optional): Theamount of NEAR
9. allowed to be spent ongas
10. . If omitted, the key can consumeunlimited
11. as gas

Function-Call keys have the main purpose of being shared, so third-parties can make contract calls in your name. This is useful inmultiple scenarios as we will see below.

tip Function-Call keys are secure to share, as they only permit calls to a specific contract and prohibit NEAR token transfers.

Benefits of Function-Call Keys

Function Call Keys allows you to providerestricted access to third parties. This key type, unique to NEAR, enables several use-cases worth discussing.

Enhancing User Experience

The most common use case forFunction-Call keys is to allow an application to sign transactions on the user's behalf.

Imagine you are developing a game that records the user's score on a smart contract. On other chains, you would have to disrupt the user's experience to request transaction signatures each time the game needs to update the score.

With NEAR, you can request the user to generate aFunction-Call key for the game's contract and share it with the game. This way, the game can sign transactions in the user's name, eliminating gameplay interruptions.

Sharing this key is safe for the user, because even in the case of somebody stealing it, they would only be able to call the score-keeping method, and nothing else.

Simple Onboarding

Another common use-case ofFunction-Call keys is to simplify theonboarding process for new users. It works as follows:

First create a contract that has a method calledcreate_account . This method should only be callable by the contract itself and, when executed, should create a new account and transfer some tokens to it.

You can then create multipleFunction-Call in the contract's account, that only allow to callcreate_account . Drop these keys to your friends, so they can call the method, and easily create an account with some tokens.

tip This is the basic principle behindNEAR Drops , a way to distribute assets to a large number of users

Key Rotation and Recovery

The presence of multiple keys allows for easyrotation andrecovery . If you suspect a key might be compromised, you can promptly remove it or replace it with a new one, similar to changing your password on a website.

You can also establish a key-recovery contract in your account and generate a "recovery key" for a trusted party. This key would only be used to initiate the recovery process.

In case of necessity, the trusted party can trigger the recovery process, assisting in the creation of a new full-access key for you.

Locked Accounts

If you remove all keys from an account, then the account will becomelocked , meaning that no external actor can perform transactions in the account's name.

In practice, this means that only the account's smart contract can transfer assets, create sub-accounts, or update its code.

Locking an account is very useful when one wants to deploy a contract, and let the community be assured that only the contract is in control of the account.

warning An account could still add keys to itself through a smart contract, effectively allowing the contract to unlock the account. Notice that this can only be done if the contract is deployed before the account is locked Edit this page Last updatedonMar 25, 2024 bygagdiez Was this page helpful? Yes No