chestnut

5/9/2019
Draft 1

Table Of Contents

chestnut

One big obstacle to widespread adoption of crypto assets is the fear users have in taking responsibility for managing and protecting their own money. Chestnut takes some of the best security practices from the traditional banking system and bridges them over to the blockchain space. With this, users can feel the same level of safety they have become accustomed to with the autonomy and transparency that the blockchain offers.

Chestnut utilizes a smart contract within a multi-sig account to enable users to set specific and customizable rules or restrictions on the activity, size or type of transactions that can be executed by the account. Similar to security and fraud protection offerings by traditional credit card companies, Chestnut users can set spending limits, whitelist/blacklist recipients, freeze the account or nominate beneficiaries in case of emergencies. With Chestnut, users have the benefit of setting their own account rules rather than rules imposed by a third party institution.

Each account transaction passes through our smart contract, and if it does not fit the set parameters, Chestnut does not sign off on the transaction as a multi-sig. At no point does Chestnut require private information and cannot make changes to a customer's account unless the customer initiates the change.

Chestnut provides the peace of mind that so many blockchain curious members of the general public desire before leaping into the world of crypto assets.

The slew of unique characteristics blockchain applications provide has attracted the interest of individuals and businesses in recent years. Despite these characteristics unlocking an entirely new way of conducting business, blockchains immutability has also severely limited the technologies adoption. Immutability forces the history of the blockchain to be irreversible creating a single source of indisputable truth as time passes. While this keeps the blockchain auditable and transparent, it is very unforgiving when a mistake occurs. Unlike a bank that can catch and reverse a fraudulent transaction, the blockchain does not care and has unfortunately allowed a multitude of mishaps to occur. Even in basic scenarios such as transferring tokens, a simple typo could cause funds to be lost forever. The result has plagued the cryptocurrency ecosystem with wrongfully lost funds ruining the livelihood of existing users and deterring potential users from participating in the new economy.

Blockchain is not able to handle adoption while people and business are at risk like this. If we wish to change this, then we must develop the financial tools required to build safe and prosperous blockchain communities.

Previous attempts at blockchain security such as ECAF have used off-chain solutions which ultimately failed due to inevitable human error. We believe if mistakes cannot be corrected, then it is best not to make them at all. Chestnut is an on-chain solution that saves you from ever making a costly mistake again.

Chestnut is implemented using the EOS.IO software and is compatible with compliant blockchains.

Given that blockchain and EOS.IO are relatively new technologies it is fair to assume that the average reader is unfamiliar with much of the EOS Mainnet design and therefore a few concepts must first be understood.

Introduction To EOS.IO

The EOS.IO software is a blockchain architecture utilizing an operating system-like construct which can host a new generation of smart contract based applications. At its core, EOS.IO facilitates permission based accounts that communicate by transmitting actions amongst each other. Accounts can handle these actions accordingly and record application state to the blockchain by running a smart contract. Accounts are permission based[1] and allow any weighted combination of other accounts and private keys to control them. This feature allows multiple entities to own a single, multi-user, account.

Smart Contracts

Smart contracts are computer programs which can run on top of any EOS.IO account allowing them to process incoming actions, evaluate business logic, and save user data to the blockchain. Often smart contracts and the account running them share the same name as to avoid confusion. For example, the account running the `chestnutmsig` contract is also named chestnutmsig.

A default EOS.IO account has two permissions, `@owner`, which can perform any action for the account, and `@active`, which can perform any action except change the owner key. The most commonly used account structure consists of an `@owner` permission private key, which is kept offline for back-up, and an `@active` permission private key which gets imported into a wallet and used to sign everyday transactions on the blockchain.

ex:
johndoe

Figure 1. A default EOS account

Even though this default configuration is perfectly fine for regular blockchain use and does give users the ability to recover a lost active key with an owner key, it still requires the unfamiliar and cumbersome process of maintaining two private keys, one offline and one in a wallet. If the owner key is lost, there is no recovery. Placing all of this risk and security on the user has rightfully instilled fear in newcomers.

To minimize the risk of lost, hacked, or stolen private keys/accounts, the vast majority of people must adopt multi-sig accounts with trusted third parties. Removing the security burden from the customer and placing it on a known and trusted third party is comparable to how banks are entrusted to handle and secure customer funds. However, the utilization of EOS.IO smart contracts and account permissions give multi-sig accounts an advantage. Foremost, multi-sig accounts do not need to hold custody of the customers' assets, and secondly, customers can knowingly trust a smart contract as a third party since the code is both open-source and publically viewable on the EOS blockchain.

Smart Account System Overview

The Chestnut smart account system needs at minimum two EOS.IO accounts; one running the `chestnutmsig` smart contract and the other(s) being smart account users. Smart account users convert their existing EOS.IO accounts from a single user account into a multi-user account requiring signatures from both the user and the chestnutmsig account to execute a valid transaction. Rules programmed into the `chestnutmsig` smart contract for approving transactions allow the chestnutmsig account to automatically determine whether or not to sign the given transaction along with the user.

Smart accounts are multi-user accounts that link themselves with the chestnutmsig account (running `chestnutmsig` smart contract). A smart account utilizes a custom account structure that requires both the chestnutmsig account and a users' private key to act as either the users' `@active` or `@owner` permission.

Smart Accounts are created by turning an accounts `@active` permission into a multi-user permission made up of both the chestnutmsig accounts permission `@smartaccount` and the users `@chestnut` permission. Whenever the user wishes to send a token transfer or any other secured "smart" transaction supported in the future, they must complete a multi-sig transaction with valid signatures from both themselves and the chestnutmsig account.

ex:
smartjohndoe

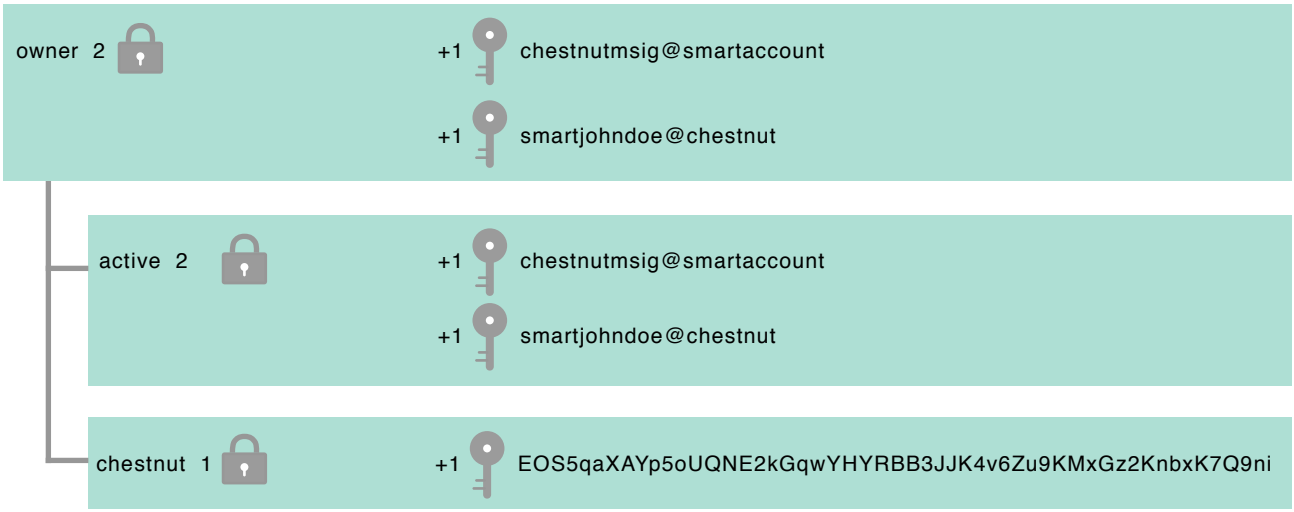


Figure 2. A Chestnut EOS smart account

In Figure-2, we assume a user owns the Chestnut smart account smartjohndoe. The account has split its active permission between its `@chestnut` permission and the `@smartaccount` permission of the chestnutmsig account. Both `smartjohndoe@chestnut` and `chestnutmsig@smartaccount` have an authority weight of 1. `smartjohndoe@active` requires an authority weight threshold of 2 to validate any transaction for `smartjohndoe@active`. The only way for this to succeed is for both the user and the chestnutmsig account permission `@smartaccount` to sign two halves of a multi-sig transaction.

A smart account has three permissions; an `@owner` permission used for recovery and beneficiaries, an `@active` permission used for smart account smart transfers, and a `@chestnut` permission used by the end user to control the account.

The `@chestnut` permission can only interact with the chestnutmsig account and can propose and approve multi-signature transactions forcing the smart account user to send multi-sig proposals to the chestnutmsig account to validate. The permission can also be expanded to allow interaction with other smart contracts (dApps) by linking the `@chestnut` permission authority with the dApps contract allowing the smart account to maintain an active whitelist of dApps they feel secure using.

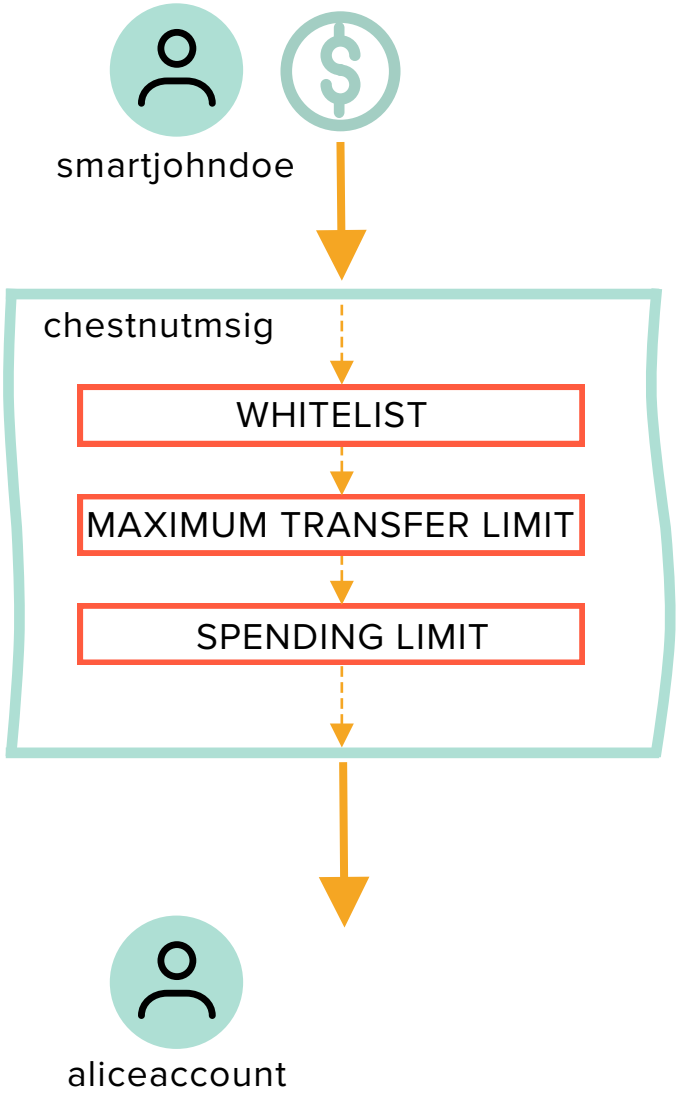
The `chestnutmsig` smart contract runs on top of the chestnutmsig account. The chestnutmsig account has a child permission to its `@active` permission called `@smartaccount` with the single authority `chestnutmsig@eosio.code`. The `@eosio.code` authority is a way of stating that the smart contract code itself has the authority sign transactions. Since this is the only authority added to `chestnutmsig@smartaccount` only the smart contract code can sign and execute the second half of smart account multi-sig transactions.

ex:

chestnutmsig

Figure 3. The chestnutmsig EOS account

Multi-signature token transfers proposed by a smart account and executed by the chestnutmsig account are called smart transfers since they add a level of brains to the transfer. Currently, smart transfers check against three user settable security parameters. Smart transfers check the recipient against a whitelist, the quantity against a transfer limit, and the total amount spent overtime against a spending limit. The whitelist confirms that the smart account trusts the recipient and that there are no errors or typos in the recipient account name. The quantity limit protects the smart account from accidentally sending an unintentionally large amount of tokens at once, and the spending limit provides fine-grain control over the smart accounts' total spending. Each of these parameters can be toggled on or off by the smart account allowing the use of specific features as need.



Smart accounts also contain a smart contract (dApp) whitelist to manage which smart contracts the smart account is comfortable using. The smart account can interact with any contract on this whitelist and automatically blocks unknown contracts for the user.

To fully understand the importance of a smart contract whitelist, one must first know that all EOS.IO smart contracts are by default permissionless allowing anyone to call them. One can see this by looking at the vast amount of unique block explorers that although have very different user interfaces all call the same contracts under the hood. Nevertheless, along with this power comes responsibility and numerous scam and phishing imitator apps have emerged. These scams look identical to the real app but instead call different malicious smart contracts on the backend tricking users into losing tokens and stealing digital property. Allowing smart accounts the ability to whitelist trusted smart contracts renders this deception obsolete and apps calling imitator contracts are automatically blocked.

It is important to note that the smart account user has full control over which smart contract accounts they whitelist leaving them liable for their own research when adding to the whitelist.

If a smart account so wishes they can create a multi-sig `@owner` permission to enable trusted account recovery using one or more beneficiary accounts. Signatures from these trusted beneficiary accounts are then used to change the smart accounts private key if it is ever lost or misplaced.

ex:

smartjohndoe

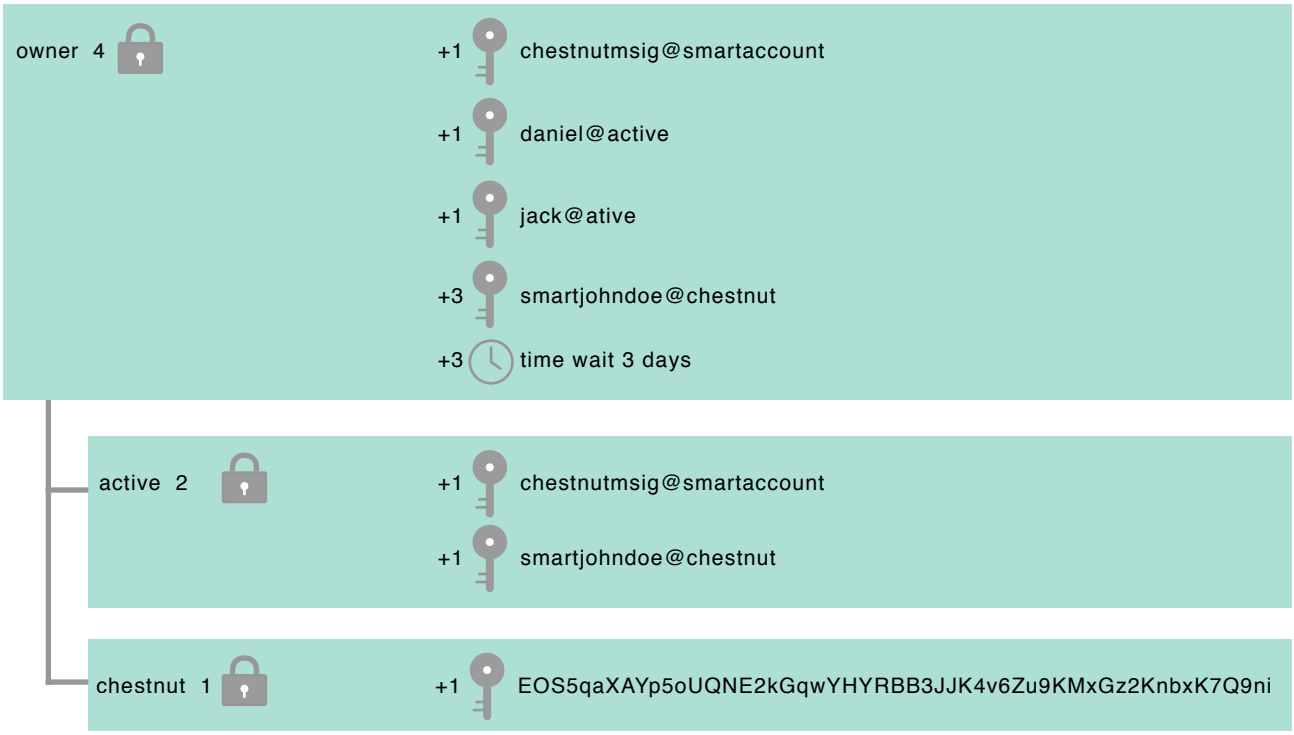


Figure 4. A Chestnut EOS smart account with recovery beneficiers

In the above example, the smart account smartjohndoe has added both jack and daniel as trusted recovery partners and beneficiaries. The `@owner` permission is required to change the `@active` or `@chestnut` authorities of a smart account. In the above configuration, the `@owner` permission has a threshold of 4, which means the weight of all authority signatures summed together must be equal or greater than 4 to be considered a valid transaction. Anything less does not pass.

smartjohndoe has lost his key `@chestnut` and needs to switch it to a new key. He needs one of his trusted beneficiaries, either **daniel** or **jack** to recovery his key.

1) **smartjohndoe** loses his key and will need either **daniel** or **jack** to for this account for him

```
+1 daniel          +1 jack
+3 wait time       +3 wait time
-------------      -------------
 4 PASS             4 PASS
```

2) The three days wait till will give **smartjohndoe** time to cancel the transaction with the chestnutmsig smart contract in the event **smartjohndoe**'s beneficiaries are acting maliciously.

```
+1 chestnutmsig@smartaccount
+3 smartjohndoe
---------------------------
 4 PASS
```

It is also possible to use an additional multi-user account composed of multiple beneficiaries to recover a smart account. Doing so would let the smart account require more than one beneficiary in a multi-sig to recover the account. The only drawback is the newly created account must be paid for and funded.

Chestnut Smart Accounts provide an additional level of security and certainty to any EOS.IO account without compromising property ownership. We envision most every day EOS.IO users utilizing the entirety of Chestnut Smart Account features to feel safe within the blockchain realm. We also see a strong use-case for businesses to adopt a similar smart account structure to keep their digital assets as safe and automated as possible.

Learn more at *http://www.chestnutaccounts.com/*

[1] See “Role Based Permission Management” in *https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md#accounts*