MultiChain permissions management

MultiChain provides full control over permissions at the network level

Permissions in MultiChain

In MultiChain, there are many global permissions that can be granted to addresses:

- `connect` – to connect to other nodes and see the blockchain's contents.
- `send` – to send funds, i.e. sign inputs of transactions.
- `receive` – to receive funds, i.e. appear in the outputs of transactions.
- `issue` – to issue assets, i.e. sign inputs of transactions which create new native assets (/developers/native-assets/).
- `create` – to create streams, i.e. sign inputs of transactions which create new streams (/developers/data-streams/).
- `mine` – to create blocks, i.e. to sign the metadata (/developers/mining-block-signatures/) of coinbase transactions.
- `activate` – to change `connect`, `send` and `receive` permissions for other users, i.e. sign transactions which change those permissions.
- `admin` – to change all permissions for other users, including `issue`, `mine`, `activate` and `admin`.

MultiChain 2.0 also supports six custom permissions that are useful for defining roles that are enforced by smart filters (/developers/smart-filters/):

- `high1`, `high2`, `high3` – custom permissions that can be set by users with `admin` privileges.
- `low1`, `low2`, `low3` – custom permissions that can be set by users with `admin` or `activate` privileges.

All permissions are assigned on a per-address basis, where addresses can either be pubkey hashes (https://bitcoin.org/en/glossary/p2pkh-address) or script hashes (https://bitcoin.org/en/glossary/p2sh-address). Depending on the `anyone-can-*` settings in the blockchain parameters (/developers/blockchain-parameters/), some operations might be completely unrestricted. Permissions can also be made temporary by limiting them to a specific range of block numbers, so that they are available only to transactions which appear within those blocks.

All permissions are granted and revoked using network transactions containing special metadata, which are easy to send using the `grant` and `revoke` commands for `multichain-cli` or the JSON-RPC API. The creator of a chain's first genesis block automatically receives all regular (non-custom) permissions. This administrator can then grant permissions to other addresses, including `admin` and `activate` permissions.

If a MultiChain wallet contains multiple public/private key pairs, when performing a restricted operation such as asset issuance or stream creation, it will automatically use one of its keys (and corresponding unspent transaction outputs) which is permitted to perform that operation. There are also `*from` APIs which allow explicit control over which address is used.

MultiChain also allows `admin`, `activate` and `write` permissions for be set for individual streams, and `admin`, `issue`, `send` and `receive` permissions to be set for individual assets (the latter two in MultiChain 2.0 only).

How permissions are applied

The `connect` permission is applied during the peer-to-peer handshaking (/developers/peer-handshaking/) that takes place between nodes, in which each node proves its ownership of the private key corresponding to a particular address. If permission to connect to the network is revoked from an address, nodes immediately disconnect any connections with nodes that used the revoked address during handshaking.

Other permissions are applied by checking if each transaction (including permission assignment transactions) is valid according to the permissions transactions which came before it. (One exception is `receive` permissions, which can be used in the same transaction in which they are assigned.) Although different nodes may see transactions in different orders, the blockchain finalizes their definitive ordering, and all transactions are 'replayed' in blockchain order by all nodes to check their validity. If a transaction in a block is disallowed according to this rule, the entire block is rendered invalid. In contrast to the white paper, the creator of a block is checked *before* applying all privilege changes defined within that block's transactions.

Permissions in regular transactions

A regular transaction, which does not create an asset or stream or assign permissions, is valid if all of its inputs and all of its outputs are permitted. Each input is permitted if one of the following conditions is true:

- If `anyone-can-send` is `true` in the blockchain parameters (/developers/blockchain-parameters/).
- For a regular pay-to-pubkey-hash signature (or a pubkey signature), the address corresponding to the pubkey must have `send` permissions.
- For a multisig input (whether pay-to-scripthash or bare multisig), the address corresponding to at least one of the pubkeys, or the pay-to-scripthash address itself, must have `send` permissions.

For regular transactions, each output is permitted if one of the following conditions is true:

- If `anyone-can-receive` is `true` in the blockchain parameters (/developers/blockchain-parameters/).
- If `anyone-can-receive-empty` is `true` in the blockchain parameters (/developers/blockchain-parameters/), and the output contains no native currency, assets or other metadata.
- For a regular pay-to-pubkey-hash script (or a pubkey script), the destination address (or address corresponding to the pubkey) must have `receive` permissions.
- For a script hash script, the scripthash address must have `receive` permissions.
- For an `m`-of-`n` bare multisig script, at least `n-m+1` addresses must have `receive` permissions. This ensures that the script cannot be redeemed only using the pubkeys of addresses which do not have `receive` permissions. If this is too restrictive for your needs, you may wish to set `anyone-can-receive` to `true` and focus on managing `send` permissions instead.
- All nulldata (`OP_RETURN` metadata) scripts are accepted.

Note however that if a transaction involves an asset with per-asset `send` or `receive` restrictions, then the appropriate input and/or output addresses must also have the appropriate per-asset permission (MultiChain 2.0 only).

Permissions for other transactions

A transaction which issues a new asset is valid if one of its inputs is signed using an address with `issue` permissions, and the signature type is `SIGHASH_ALL`, meaning that the entire transaction was signed. These addresses automatically receive `admin` and `issue` permissions for the new asset. A transaction which performs a follow-on issuance for a particular asset is valid if one of its inputs is signed using an address which has `issue` permissions for that asset, and if the input's signature type is `SIGHASH_ALL`.

A transaction which creates a new stream is valid if the output creating the stream is covered by the signature of one or more addresses with `create` permissions. These addresses automatically receive `admin`, `activate` and `write` permissions for the new stream. See the data streams (/developers/data-streams/) page for information about how write permissions in a stream are applied.

In MultiChain 2.0, transactions performing other special operations are valid if the output performing that operation is covered by the signature of one or more addresses with the following permissions:

- Creating an upgrade: `admin` and `create` required.
- Creating a transaction filter: `admin` and `create` required.
- Creating a stream filter: `create` permissions.
- Approving or disapproving an upgrade: `admin` permissions.
- Approving or disapproving a transaction filter: `admin` permissions.
- Approving or disapproving a stream filter for a stream: per-stream `admin` permissions.

A coinbase transaction (embedded by the block's creator) is valid if the output address has either `receive` or `mine` permissions. Separately from this, the block creator proves their identity by signing the block (/developers/mining-block-signatures/) inside the coinbase transaction using an address with `mine` permissions.

A transaction which assigns a global permission is valid if, for every output in which `connect`, `send`, `receive` or `low1|2|3` permissions were assigned, an address with `admin` or `activate` permissions signed the content of that output. For outputs assigning `create`, `issue`, `mine`, `activate`, `admin` or `high1|2|3` permissions, only `admin` signatures are accepted. In either case, the change is accepted if a sufficiently privileged user signed any input with the signature type `SIGHASH_ALL`, or if they signed the input with the same index as that output with `SIGHASH_SINGLE`. Some additional notes:

- If `support-miner-precheck` is `true` in the blockchain parameters (/developers/blockchain-parameters/), an administrator can only make an `admin` or `mine` permissions change in a transaction if the transaction contains a cache of the `scriptPubKey` of an output spent in one of that administrator's inputs. This "input cache" will allow future versions of MultiChain to check the mining permissions of blocks on a fork without switching to that fork.
- Outputs which only assign permissions can contain zero units of native currency, even if `minimum-per-output` in the blockchain parameters (/developers/blockchain-parameters/) is greater than zero. However any quantity more than zero must satisfy the `minimum-per-output` constraint.
- Contradicting permissions assignments within a single transaction are applied chronologically in the order in which they appear, i.e. only the last assignment counts.

- Depending on the blockchain parameters, changes to `admin`, `activate`, `mine`, `issue` and `create` permissions may require consensus between multiple administrators. If the signatures of multiple administrators cover the output which affects one of these permissions, then this is counted as a vote by all of those administrators.
- If an address is granted administrator privileges within a transaction, those privileges cannot be used within the same transaction to affect the permissions of other users.

All of these rules can be bypassed by setting the appropriate `anyone-can-*` value in the blockchain parameters (/developers/blockchain-parameters/) to `true`.

A transaction which assigns a per-asset or per-stream permission is valid if, for every output in which permissions were assigned, an address with `admin` permissions for that entity signed the content of that output. For outputs assigning `write` permissions for a stream, or `send` or `receive` permissions for an asset, an address with per-entity `activate` permissions is sufficient.

Implied permissions

Several higher-level permissions automatically imply some lower-level permissions. This avoids illogical situations where (for example) an address has been granted permission to issue assets, but it cannot actually do so because it does not have permission to sign the transactions in which assets are issued. Below is a full list of such implied permissions:

- `admin` → `activate`, `send`, `receive`, `connect`
- `activate` → `send`, `receive`, `connect`
- `mine` → `connect`, `receive` (in coinbase transaction only)
- `issue` → `send`
- `create` → `send`

Permissions consensus

Depending on the blockchain parameters (/developers/blockchain-parameters/), a single administrator may not be able to grant or revoke `admin`, `activate`, `mine`, `issue` or `create` permissions on their own. Instead, a number of administrators are required to agree on the same change, and only then is it applied. This ensures that a single rogue administrator is not able to unilaterally decide who has the right to perform the blockchain's most critical tasks.

The number of administrators required to change the administrator privileges of an address is calculated by multiplying the current number of active administrators by the `admin-consensus-admin` value in the blockchain parameters, then rounding up. The number required to change the other privileges is calculated in a similar way, but substituting the `admin-consensus-activate`, `admin-consensus-mine`, `admin-consensus-issue` or `admin-consensus-create` value. However, if the blockchain is in its first `setup-first-blocks`, no consensus is required, i.e. these parameters are effectively considered to be `0`.

Note that consensus is only achieved if sufficient administrators have indicated the exact same start block and end block for a particular address and privilege. If an administrator has more than one outstanding assignment of block ranges for a particular privilege and address, the assignment with the highest timestamp (as embedded in the transaction assigning permissions) is interpreted as that administrator's latest vote. Once consensus is successfully achieved for a particular permission and address, all previous votes are no longer relevant.

Permissions in transaction data

For regular use of MultiChain, you can ignore the technical details below. They are only relevant if you want to work with the raw data within MultiChain transactions. Note that you can also use the raw transactions (/developers/raw-transactions/) APIs to encode and decode this information.

If an output contains `OP_DROP` permissions metadata and is covered by the signature of an appropriately privileged user in one of the inputs, permissions are assigned to the address represented by that output, whether a pubkey-hash or script-hash. In the case of a pay-to-pubkey output, permissions are assigned to the address corresponding to that public key. Note that changes to `admin`, `activate`, `mine`, `issue` or `create` permissions may require further assignments by other administrators in order to reach the necessary level of consensus (see above).

Global permissions metadata

Global permissions assignments for the address in a transaction output are encoded before an `OP_DROP` opcode (`0x75` in hex) as follows:

Field	**Size**	**Description**
Identifier	4 bytes	`spkp` or `0x73 0x70 0x6b 0x70`
Permissions	4 bytes	Bitmap of permissions assigned, where `connect=1`, `send=2`, `receive=4`, `issue=16`, `create=32`, `mine=256`, `low1=512`, `low2=1024`, `low3=2048`, `admin=4096`, `activate=8192`, `high1=131072`, `high2=262144`, `high3=524288`. The resulting unsigned 32-bit integer is written in small-endian order.
Start block	4 bytes	Block number in which permission starts, unsigned 32-bit integer in small-endian order.
End block	4 bytes	Block number in which permission ends, unsigned 32-bit integer in small-endian order.
Timestamp	4 bytes	A monotonically increasing timestamp, unsigned 32-bit integer in small-endian order. This helps resolve conflicts between permissions assigned by the same administrator, where the permission is subject to consensus (see below).

A permission is granted permanently by setting the start block to `0` and the end block to `4294967295`, the maximum 32-bit unsigned integer. A permission is revoked by setting both the start block and end block to `0`.

Per-asset and per-stream permissions

Per-entity permissions assignments for the address in a transaction output are encoded using the following structure:

```
entity-identifier OP_DROP entity-permissions OP_DROP
```

The `entity-identifier` has the following structure:

Field	**Size**	**Description**
Prefix	4 bytes	`spke` or `0x73 0x70 0x6b 0x65`
Entity	16 bytes	First 16 bytes of asset's first issuance txid or stream creation txid in reverse order.

The `entity-permissions` uses the same structure as global permissions changes:

Field	**Size**	**Description**

Identifier	4 bytes	`spkp` or `0x73 0x70 0x6b 0x70`
Permissions	4 bytes	Bitmap of permissions assigned, where `send=2` (assets only), `receive=4` (assets only), `write=8` (streams only), `issue=16` (assets only), `admin=4096` (assets or streams), `activate=8192` (streams only). The resulting unsigned 32-bit integer is written in small-endian order.
Start block	4 bytes	Block number in which permission starts, unsigned 32-bit integer in small-endian order.
End block	4 bytes	Block number in which permission ends, unsigned 32-bit integer in small-endian order.
Timestamp	4 bytes	A monotonically increasing timestamp, unsigned 32-bit integer in small-endian order.

A permission is granted permanently by setting the start block to `0` and the end block to `4294967295`, the maximum 32-bit unsigned integer. A permission is revoked by setting both the start block and end block to `0`.

Transaction input cache

An input cache (for `support-miner-precheck`) is represented in a separate output as below, followed by an `OP_DROP` (`0x75`) and `OP_RETURN` (`0x6a`):

Field	**Size**	**Description**
Identifier	4 bytes	`spki` or `0x73 0x70 0x6b 0x69`
Repeat the below for each input whose previous transaction output is cached		
Input index	4 bytes	The index of the input, as an unsigned 32-bit integer in small-endian order.
Length	1-9 bytes	Bitcoin-style variable-length integer (https://en.bitcoin.it/wiki/Protocol_documentation#Variable_length_integer) indicating the length of the cached script in bytes.
Script	Variable	The cached script (`scriptPubKey`) of the spent output, in raw binary.