

# EOS 权限系统与签名认证

Frank

EOSLaoMao

May 12, 2019



# Outline

Permission Introduction

Transaction Generation

Transaction Verification

# Permission

```
$ cleos set account permission [account] [permission] [authority]  
[parent]
```

```
cleos set account permission alice transfer  
↳ '{"threshold":1,"keys":[], "accounts":[]}' "active" -p  
↳ alice@active
```

```
alice@  
└─ owner  
   └─ active
```

```
alice@  
└─ owner  
   └─ active  
      └─ transfer
```

# Authority

- ▶ Threshold
- ▶ Keys
- ▶ Accounts
- ▶ Waits

active@alice

```
{
  "threshold": 2,
  "keys": [
    {
      "key": "EOS6fzbrDV6qL99HatqAKM2dSdUVQmiDsZsD2K58YQ3MXL4CBphN5",
      "weight": 1
    },
    {
      "key": "EOS7VJkEQhQw3JQNGhGwbX3oH3AEZExia5cTZzSH1fm3y7HNVueck",
      "weight": 1
    }
  ],
  "accounts": [
    {
      "permission": {
        "actor": "bob",
        "permission": "eosio.code"
      },
      "weight": 2
    }
  ],
  "waits": [
    {
      "wait_sec": "3600",
      "weight": 1
    }
  ]
}
```

# Permission Link

Accounts and Permissions are stored in the chain state database.

See: [bytemaster/ChainBase](#)

```
$ cleos set action permission [account] [code] [type] [requirement]
```

account_object
name
...
...

permission_object
name
parent
owner
authority
...

permission_link_object
account
code
message_type
required_permission
...

# Push Transaction

使用 eosjs 和 eosjs-ecc 这两个项目来分析整个签名过程。现在我们要发送一个包含如下 action 的 transaction。完整代码：[sign-trx-demo/index.js](https://github.com/steemit/sign-trx-demo/blob/master/index.js)

```
const action = {
  account: 'eosio',
  name: 'newaccount',
  authorization: [{
    actor: 'eosio',
    permission: 'active',
  }],
  data: {
    creator: 'eosio',
    name: 'alice',
    owner: {
      threshold: 1,
      keys: [{
        key: 'EOS6MRyAjQq8ud7hVNYcfnVPJqcVpscN5So8BhtHuGYqET5GDW5CV',
        weight: 1
      }],
      accounts: [],
      waits: []
    },
    active: {
      threshold: 1,
      keys: [{
        key: 'EOS6MRyAjQq8ud7hVNYcfnVPJqcVpscN5So8BhtHuGYqET5GDW5CV',
        weight: 1
      }],
      accounts: [],
      waits: []
    },
  },
};
```

# Procedure Source Code

1. `eosjs/src/eosjs-api.ts:214 transact()`
2. `eos/src/eosjs-jssig.ts:33 sign()`
3. `eosjs-ecc/src/signature:117 Signature.sign()`
4. `eosjs-ecc/src/signature.js:195 Signature.signHash()`
5. `eosjs-ecc/src/signature.js:195 Signature.toBuffer()`
6. `eosjs-ecc/src/signature.js:195 Signature.toString()`
7. `eosjs-ecc/src/key_utils.js:181 checkEncode()`

# Pseudo Code

```
1  action.data = abi.serialize( action.data )
2  transaction = { ..., actions }
3  digest = sha256( concat( chain_id, serialize(transaction), buffer[32] ) )
4  signature = concat( i, ecc.encrypt( digest, private_key ) )
5  checksum = ripemd160( concat(signature, type) )
6  text = concat( 'SIG_K1_', base58.encode( concat( signature, checksum ) ) )
```



# Output Log

transaction

```
{
  "expiration": "2019-05-09T19:08:13.000",
  "ref_block_num": 12516,
  "ref_block_prefix": 3102567877,
  "actions": [
    {
      "account": "eosio",
      "name": "newaccount",
      "authorization": [
        {
          "actor": "eosio",
          "permission": "active"
        }
      ],
      "data": "0000000000EA30550000000000855C3401000000010002C0DED2BC1F1305FB0FAAC5E6C03EE3A1924
↪      234985427B6167CA569D13DF435CF0100000001000000010002C0DED2BC1F1305FB0FAAC5E6C03EE3
↪      A1924234985427B6167CA569D13DF435CF01000000"
    }
  ]
}
```

digest

c4822b63ba2960e4bab2700b201b525af173744f911de5d0ff4ce5bac11da070

text

SIG\_K1\_KfqCE8YMxNkdC79ihnFejau37XkVMELpmtfuJPGdS4aTN5ypmb9iXyPsAFVZ8akJC2uvXxq5M8jFqmfoFV51Wvpba55d85

# Public Key Recovery

Public key can be recovered from original content, ECDSA signature and extra public key information.

```
public_key = recover( signature, digest )
```

[eos/libraries/chain/transaction.cpp:98](#) `transaction::get_signature_keys()`

```
1 recovered_pub_keys.clear();
2 const digest_type digest = sig_digest(chain_id, cfd);
3 // c4822b63ba2960e4bab2700b201b525af173744f911de5d0ff4ce5bac11da070
4 std::unique_lock<std::mutex> lock(cache_mtx, std::defer_lock);
5 fc::microseconds sig_cpu_usage;
6 for(const signature_type& sig : signatures) {
7     auto now = fc::time_point::now();
8     EOS_ASSERT( now < deadline, tx_cpu_usage_exceeded, "transaction signature verification executed for too
9         ("now", now)("deadline", deadline)("start", start) );
10    public_key_type recov;
11    const auto& tid = id();
12    lock.lock();
13    recovery_cache_type::index<by_sig>::type::iterator it = recovery_cache.get<by_sig>().find( sig );
14    if( it == recovery_cache.get<by_sig>().end() || it->trx_id != tid ) {
15        lock.unlock();
16        recov = public_key_type( sig, digest );
17        fc::microseconds cpu_usage = fc::time_point::now() - start;
18        lock.lock();
19        recovery_cache.emplace_back( cached_pub_key{tid, recov, sig, cpu_usage} ); //could fail on dup signature
20        sig_cpu_usage += cpu_usage;
21    } else {
22        recov = it->pub_key;
23        sig_cpu_usage += it->cpu_usage;
24    }
25    lock.unlock();
26    ...
27 }
```

# Multi-threaded recovery

Release 1.5.0 introduces the support for recovering keys from cryptographic signatures (aka signature verification) for transactions and blocks across multiple threads.

[eos/libraries/chain/transaction\\_metadata.cpp:28](https://eos.io/libraries/chain/transaction_metadata.cpp#L28)

```
1  signing_keys_future_type transaction_metadata::start_recover_keys( const
   ↪   transaction_metadata_ptr& mtrx, boost::asio::thread_pool& thread_pool, const
   ↪   chain_id_type& chain_id, fc::microseconds time_limit )
2  {
3      if( mtrx->signing_keys_future.valid() && std::get<0>( mtrx->signing_keys_future.get() ) ==
   ↪       chain_id ) // already created
4          return mtrx->signing_keys_future;
5
6      std::weak_ptr<transaction_metadata> mtrx_wp = mtrx;
7      mtrx->signing_keys_future = async_thread_pool( thread_pool, [time_limit, chain_id, mtrx_wp]()
   ↪       {
8          fc::time_point deadline = time_limit == fc::microseconds::maximum() ?
9              fc::time_point::maximum() : fc::time_point::now() + time_limit;
10         auto mtrx = mtrx_wp.lock();
11         fc::microseconds cpu_usage;
12         flat_set<public_key_type> recovered_pub_keys;
13         if( mtrx ) {
14             const signed_transaction& trn = mtrx->packed_trx->get_signed_transaction();
15             cpu_usage = trn.get_signature_keys( chain_id, deadline, recovered_pub_keys );
16         }
17         return std::make_tuple( chain_id, cpu_usage, std::move( recovered_pub_keys ) );
18     } );
19
20     return mtrx->signing_keys_future;
21 }
```

# Permission Verification

[eos/libraries/chain/authorization\\_manager.cpp:476 check\\_authorization\(\)](#)  
[eos/libraries/chain/include/eosio/chain/permission\\_object.hpp:51 satisfies\(\)](#)

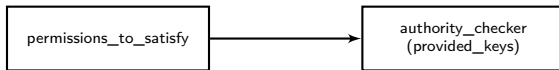
```
cleos set account permission eosio newacc '{"threshold":1,"keys":[{"key":"EOS6MRyAjQq8ud7hVNYJ  
↳ cfnVPJqcVpscN5So8BhtHuGYqET5GDW5CV","weight":1}], "accounts":[],"waits":[]}' "active"  
↳ -p eosio@active  
  
cleos set action permission eosio eosio newaccount newacc -p eosio@active
```

```
{  
  "actions": [  
    {  
      "account": "eosio",  
      "name": "newaccount",  
      "authorization": [  
        {  
          "actor": "eosio",  
          "permission": "active"  
        }  
      ],  
      "data": "..."  
    }  
  ]  
}
```

```
eosio@  
├── owner  
│   ├── active  
│       └── newacc.. min permission
```

# Authorization Verification

eos/libraries/chain/authorization\_manager.cpp:508 check\_authorization()  
eos/libraries/chain/include/eosio/chain/authority\_checker.hpp:L60 authority\_checker  
eos/libraries/chain/include/eosio/chain/authority\_checker.hpp:182 satisfied()  
eos/libraries/chain/include/eosio/chain/authority\_checker.hpp:206  
weight\_tally\_visitor



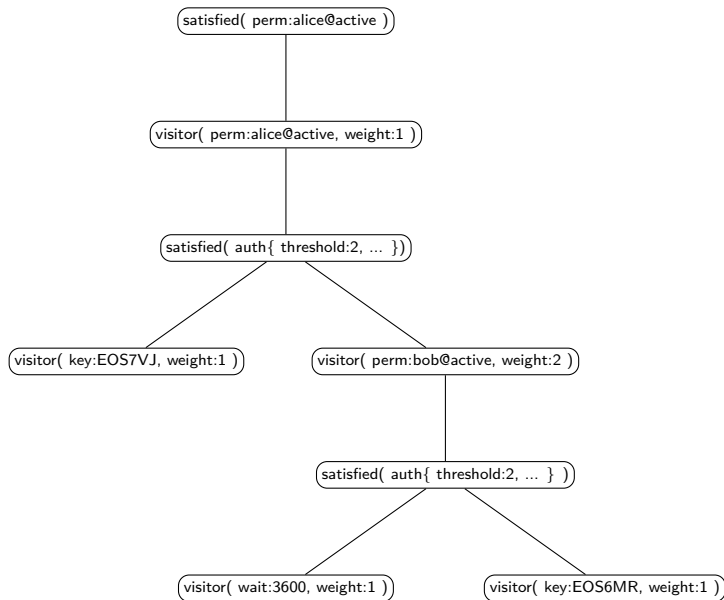
alice@active

```
{
  "threshold": 2,
  "keys": [
    {
      "key": "EOS7VJkEQhQw3JQNGhGwbX3oH3AEZE ]
      ↪ xia5cTZzSH1fm3y7HNVueck",
      "weight": 1
    }
  ],
  "accounts": [
    {
      "permission": {
        "actor": "bob",
        "permission": "active"
      },
      "weight": 2
    }
  ],
  "waits": []
}
```

bob@active

```
{
  "threshold": 2,
  "keys": [
    {
      "key": "EOS6MRyAjQq8ud7hVNYcfnVPJqcVps ]
      ↪ cN5So8BhtHuGYqET5GDW5CV",
      "weight": 1
    }
  ],
  "accounts": [],
  "waits": [
    {
      "wait_sec": "3600",
      "weight": 1
    }
  ]
}
```

# Authorization Verification



# Build & Debug

Build debug version nodeos.

```
./scripts/eosio_build.sh -o Debug
```

Run single node with gdb.

```
gdb ./build/programs/nodeos/nodeos
```

```
(gdb) run -e -p eosio --plugin eosio::chain_api_plugin --data-dir=data
```

launch.json for vscode: [launch.json](#)

# References

<https://developers.eos.io/eosio-cleos/docs/adding-eosio-code-to-active-authority-with-cleos-helper>  
<https://developers.eos.io/eosio-nodeos/docs/accounts-and-permissions>  
<https://medium.com/@eoscafeblock/how-permissions-work-in-eos-82e55eb1d896>  
<https://medium.com/leclevietnam/understanding-eos-permission-ee60dcfec8ad>  
<https://github.com/EOSIO/eos/tree/448287d520f5f8c282139ecc60f227c218d42e82>  
<https://crypto.stackexchange.com/questions/18105/how-does-recovering-the-public-key-from-an-ecdsa-signature-work>