MODCHAIN

”A retro-cyberpunk architecture for modular computation, consensus, and crypto-economics”

Module

In the **ModChain** architecture, a module is an autonomous programmable unit—an execution node that owns its memory, exposes logic, and operates independently within a modular network.

Each module maintains internal **state** (a persistent variable store), and defines executable **functions**—which may be deterministic or probabilistic. These functions are triggered through client-signed JSON-RPC calls, interacting securely via public-key cryptography.

Every module is cryptographically identified using keys compatible with `sr25519` or `ecdsa`. Modules can sign, verify, encrypt, and decrypt data independently.

Key Generation

To generate a key:

```
c key fam
```

Example output:

```
<Key(address=5Gs51y... crypto_type=sr25519)>
```

ModChain supports multiple crypto types like `sr25519` (DOT) and `ecdsa` (ETH), with modularity to support more.

To invoke a function:

```
c fn **params            # for root modules
c module/fn **params     # for nested modules
```

Consensus Mechanisms

Each module is governed by a **consensus module**, which handles economic trust, staking, dispute resolution, and transaction verification.

Consensus 0 – Proof of Interaction

Clients stake tokens for at least 1 epoch. When making a transaction, the stake is locked and deducted proportionally per use. Servers batch transactions and post them to chain periodically.

Consensus X – Custom Consensus (ZK / Interop)

Future modules may support zk-proofs or interchain settlement mechanisms, maintaining modularity and application-specific logic.

Server Layer

A **Server** is an HTTP/WS process that exposes module functions over a JSON-RPC interface. Functions must be explicitly whitelisted.

Start a server:

```
c serve api
```

Query API:

```
c call api
```

By default, servers expose ports within `50050-50150`.

Clients & Auth

Clients interact via signed requests.

Request structure

```
{
  "url": "ip:port/fn",
  "params": { "query": "whatsup" }
}
```

Auth blob

```
{
  "module": "<module_key>",
  "fn": "function_name",
  "params": {...},
  "time": "<utc>",
  "max_usage": 1.0
}
```

Generate headers:

```
c auth/generate auth_data     headers
```

Headers format

```
{
  "data": "sha256(auth_data)",
  "key": "<client_address>",
  "signature": "<sig>",
  "time": "<utc>",
  "max_usage": 1.0
}
```

Transactions

Each function execution returns a **transaction receipt**:

```
{
  "fn": "fn_name",
  "params": {...},
  "cost": 0.123,
  "result": {...},
  "client": {header...},
  "server": "<signature>"
}
```

Receipts are batched offchain and posted onchain at epoch close. Stakes are reconciled between client and server.

Disputes

If either side cheats, they enter arbitration. Both client and server lock liquidity. A quorum of N random validators resolve the case. Loser forfeits liquidity to the accuser and validators.

Cost Governance

- **Server** defines cost-per-call

- **Client** defines `max`$_u sage$ This two-sided constraint model prevents abuse and guarantees predictability.

Network (Nets)

Modules can link into **graphs** or **trees**, forming permissioned or trustless networks onchain or offchain.

Links

Links define directional relationships with optional metadata:

```
c link {parent_key} {child_key} profit_share=5 data={
    info_or_ipfs_hash}
```

```
[Parent Module]
      |
 [Link: 20%]
      |
[Child Module]
```

Topologies

Supported topologies:

- **Replica Sets**: homogeneous children
- **Subnets**: heterogeneous competition
- **Recursive Trees**: arbitrary hierarchy

Links may reference keys off-network, verified by parent only.

Version Control (ModChain)

ModChain introduces decentralized module versioning inspired by git—but simpler.

Each module folder is hashed into a **CID tree** (e.g. SHA-256) with one-depth path maps.

```
c update <module> mode=ipfs|s3|arweave
```

Creates:

```
{
  "data": {path_to_file: content},
  "previous_uri": [prev_versions...]
}
```

This is a **modchain**—a version history where a single actor or multisig group can commit updates.

License