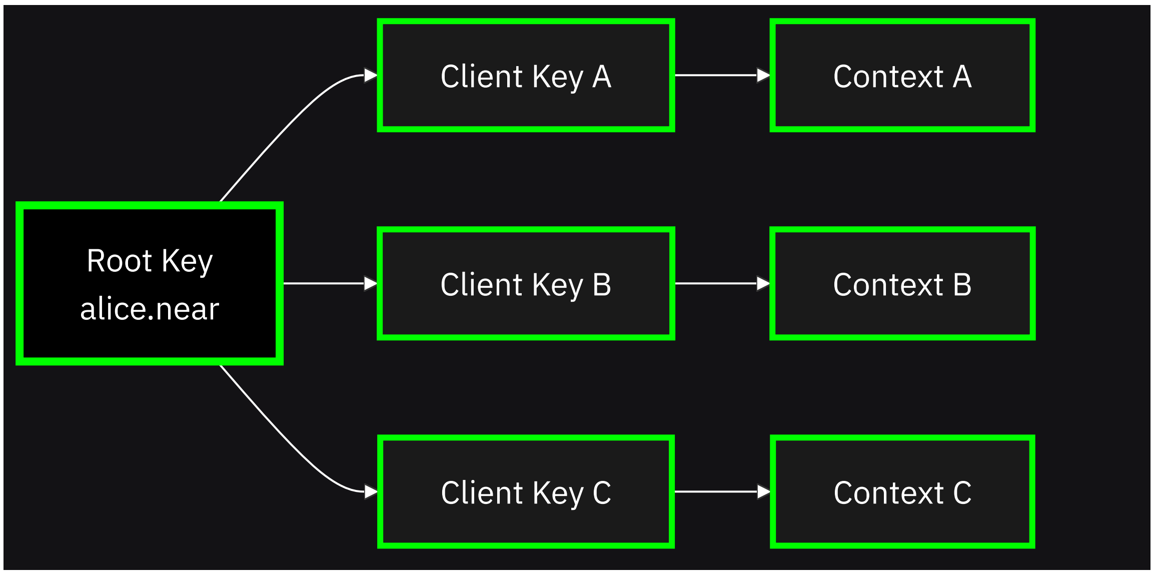
# Identity

Calimero uses **cryptographic identities** to manage access control and authentication across the network. Each participant has one or more identities that prove ownership and grant permissions.

## Identity Model

Calimero supports a hierarchical identity model:



### Root Keys

A **root key** is an authentication credential that represents a user's master identity in the Calimero auth system. It's typically:

- Generated from a NEAR wallet or from username / password combination
- Used for high-level operations (creating contexts, managing memberships)
- Stored securely (hardware wallet, keychain, etc.)

### Client Keys

**Client keys** are derived from root keys and used for:

- Executing methods in specific contexts
- Signing transactions and deltas

- Proving membership in contexts

**Benefits:**

- **Isolation**: Compromise of one client key doesn't affect others
- **Revocation**: Can revoke access per-context without changing root key
- **Privacy**: Different keys for different contexts

## Identity Generation

Generate identities with `meroctl`:

```
$: meroctl --node node1 context identity generate
> +-----------------------------------------+---------------------------------
------------+
> | Context Identity Generated              | Public Key
|
>
+============================================================================
==========+
> | Successfully generated context identity |
8XG254iKm6YGNJANbkKQpFknmE27TykArAvfJPqHBmw |
> +-----------------------------------------+---------------------------------
------------+
```

See `core/crates/meroctl/README.md` for CLI details.

## Blockchain Wallet Integration

Calimero supports wallet-based authentication:

| Protocol | Identity Source |
|---|---|
| **NEAR** | NEAR account ID + signature |

**Flow:** 1. User connects wallet 2. Signs challenge message 3. Calimero verifies signature 4. JWT token issued

See `calimero-client-js/README.md` for client authentication examples.

## Authentication Flows

For wallet authentication examples, see: - **JavaScript**: `calimero-client-js/README.md` - Client-side auth flows - **Python**: `calimero-client-py/README.md` - Python client auth

## JWT Tokens

After authentication, Calimero issues JWT tokens containing: - `context_id` - Target context - `executor_public_key` - Client key for execution - `permissions` - Access permissions - `exp` - Expiration timestamp

**Usage:** - Include in API requests: `Authorization: Bearer <token>` - Tokens expire and can be refreshed - See `core/crates/auth/README.md` for details

## Key Management

**Hierarchical structure:** - Root keys delegate to client keys per context - Each context has separate client keys - Keys can be revoked independently

**Revoke access:**

```
$: meroctl --node <NODE_ID> context identity revoke <MEMBER_ALIAS>
<CAPABILITY> --as <REVOKER_ALIAS> --context <CONTEXT_ALIAS>
```

See `core/crates/meroctl/README.md` for key management commands.

**What happens:** - Key is removed from context membership - Key can no longer sign transactions for that context - Existing transactions remain valid (immutable history) - Root key remains unaffected - Removed member stops receiving updates

## Wallet Adapters

Calimero provides wallet adapters for easy integration:

### JavaScript Client

```
import {
  CalimeroConnectButton,
} from "@calimero-network/calimero-client";

// Automatically handles node connection and authentication
<CalimeroConnectButton />
```

**Supported wallets:** - NEAR Wallet

### Python Client

```python
from calimero_client_py import create_connection, create_client

# Connect to Calimero network
connection = create_connection(
    api_url="https://node.calimero.network",
    node_name="your-node-name"  # Optional but recommended for token caching
)

# Create a client from the connection
client = create_client(connection)
...
```

## Best Practices

1. **Use Client Keys**: Don't use root keys directly for context operations
2. **Rotate Keys**: Periodically rotate client keys for security
3. **Secure Storage**: Store private keys in secure keychains, never in code
4. **Multi-Sig Support**: Use multi-signature wallets for high-value contexts
5. **Key Backup**: Backup root keys securely (hardware wallet, paper backup)

## Deep Dives

For detailed identity documentation:

- **Identity Contracts**: `contracts` README - Smart contract implementations
- **Auth Service**: `core/crates/auth/README.md` - Authentication service
- **Client SDKs**: Tools & APIs - Wallet integration guides

## Related Topics

- Contexts - Where identities are used
- Applications - What identities can access
- Architecture Overview - How identity fits into the system