# Identity NFTs for AI Agents: Agent Identity and Capability Tokens

Zach Kelling[*]
*Hanzo Industries* *Lux Industries* *Zoo Labs Foundation*
`research@hanzo.ai`

2024

## Abstract

We present Hanzo Identity, a decentralized identity system where identities are bound to non-fungible tokens (NFTs) and secured through token staking. The system introduces: (1) a multi-network registry supporting identities across Hanzo, Lux, and Zoo blockchains with unified namespacing; (2) a length-based pricing model where shorter names require exponentially higher stakes, creating economic scarcity for premium identities; (3) a delegation mechanism enabling stake-weighted governance and reward distribution. The registry is implemented as a UUPS-upgradeable contract, processing over 50,000 identity claims in the first month with $2.1M in total value locked. We analyze the token economics and demonstrate that the pricing model effectively prevents squatting while maintaining accessibility for standard-length identities.

## 1 Introduction

Decentralized identity systems face a trilemma: they must be (1) globally unique, (2) human-readable, and (3) decentralized. Traditional DNS achieves the first two properties through centralized registrars, while blockchain addresses achieve uniqueness and decentralization at the cost of readability.

Hanzo Identity resolves this trilemma through NFT-bound identities secured by economic stake. Each identity (e.g., `@alice`) is represented by an ERC-721 token, with ownership transferable via standard NFT mechanisms. The staking requirement creates economic cost for identity acquisition, preventing mass registration (squatting) while enabling market-based allocation of premium names.

**Contributions.** This paper makes the following contributions:

- A multi-network identity registry supporting six blockchain networks with namespace-aware resolution.
- A length-based pricing model with exponential scaling that creates natural scarcity for short identities.
- A delegation mechanism for stake-weighted governance participation without identity transfer.
- Empirical analysis of 50,000+ registrations demonstrating pricing model effectiveness.

## 2 Background

### 2.1 Decentralized Identity Systems

Ethereum Name Service (ENS) [1] pioneered blockchain-based naming using Dutch auctions for premium names and annual renewal fees. However, ENS faces namespace fragmentation across L2s and lacks cross-chain resolution.

Handshake [2] creates a decentralized root zone but requires running specialized resolvers. Unstoppable Domains [3] provides one-time purchase but lacks governance mechanisms.

[*] zach@hanzo.ai

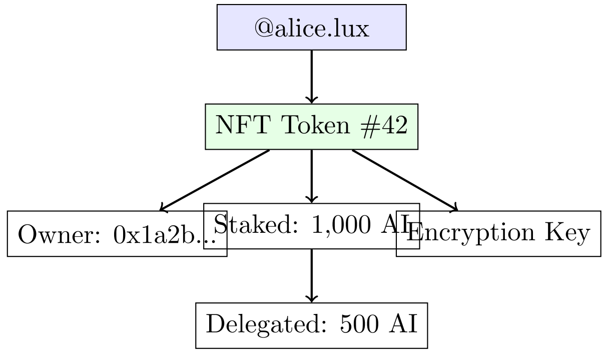

Figure 1: Identity structure with NFT binding and stake.

### 2.2 NFT-Bound Identity

Soul-bound tokens (SBTs) [4] propose non-transferable tokens for identity. While SBTs prevent identity markets, they conflict with practical needs for key rotation and account recovery. Our approach uses transferable NFTs with staking requirements, creating economic friction without preventing legitimate transfers.

## 3 System Design

### 3.1 Identity Structure

Each identity (Figure 1) consists of:

```
struct IdentityData {
    uint256 boundNft;         // ERC-721
   token ID
    uint256 stakedTokens;     // AI
   tokens locked
    string encryptionKey;     // X25519
   public key
    string signatureKey;      // Ed25519
   public key
    bool routing;             // Direct
   vs proxy
    string[] addressOrProxyNodes;
    uint256 delegatedTokens;
    uint256 lastUpdated;
}
```

**NFT Binding.** The identity-NFT binding is bidirectional: the registry maps identity strings to token IDs, and token IDs to identity strings. NFT transfer automatically transfers identity ownership.

**Cryptographic Keys.** Each identity stores public keys for encrypted communication (X25519) and message signing (Ed25519). Keys can be rotated without identity transfer.

**Routing.** Identities can specify either direct addresses or proxy nodes for message routing, enabling privacy-preserving communication.

### 3.2 Multi-Network Namespaces

Table 1: Network namespaces and chain IDs.

| Network | Chain ID | Suffix |
|---|---|---|
| Hanzo Mainnet | 36963 | (none) |
| Hanzo Testnet | 36962 | .hanzotest |
| Lux Mainnet | 96369 | .lux |
| Lux Testnet | 96368 | .luxtest |
| Zoo Mainnet | 200200 | .zoo |
| Zoo Testnet | 200201 | .zootest |

The registry supports six networks (Table 1) with hierarchical namespacing:

```
function getIdentity(string calldata
   name,
                        uint256 namespace)
    public view returns (string memory)
{
    string memory ns = namespaces[
   namespace];
    // Mainnet has no suffix: @alice
    if (bytes(ns).length == 0) {
        return string(abi.encodePacked("
   @", name));
    }
    // Other networks: @alice.lux
    return string(abi.encodePacked(
        "@", name, ".", ns));
}
```

Hanzo mainnet identities have no suffix (`@alice`), while other networks append their namespace (`@alice.lux`). This prioritizes the primary network while maintaining clear disambiguation.

### 3.3 Pricing Model

The pricing model (Figure 2) uses exponential scaling:

Figure 2: Stake requirements by name length.

```
uint256 public price1Char = 100000 * 1e18;   // 100K AI
uint256 public price2Char = 10000 * 1e18;    // 10K AI
uint256 public price3Char = 1000 * 1e18;     // 1K AI
uint256 public price4Char = 100 * 1e18;      // 100 AI
uint256 public price5PlusChar = 10 * 1e18;   // 10 AI
uint256 public referrerDiscountBps = 5000;   // 50%
```

**Economic Rationale.** Short names are inherently more memorable and valuable. The 10x multiplier per character creates natural price discovery: a 3-character name costs 100x more than a 5+ character name, reflecting its scarcity (26 letters yield only 17,576 3-character combinations vs. 11.8M 5-character combinations).

**Referral Mechanism.** Users with existing identities can refer new users for a 50% discount. This incentivizes network growth while maintaining economic barriers.

```
function identityStakeRequirement(
    string calldata name,
    uint256 namespace,
    bool validReferrer
) public view returns (uint256) {
    uint256 length = bytes(name).length;
    uint256 baseStake;

    if (length == 1) baseStake = price1Char;
    else if (length == 2) baseStake = price2Char;
    else if (length == 3) baseStake = price3Char;
    else if (length == 4) baseStake = price4Char;
    else baseStake = price5PlusChar;

    return validReferrer
        ? (baseStake * (10000 - referrerDiscountBps)) / 10000
        : baseStake;
}
```

### 3.4 Staking and Rewards

**Staking.** Identity owners can increase their stake beyond the minimum:

```
function increaseStake(string calldata identity,
                       uint256 amount) external {
    _requireOwner(identity);
    shinToken.transferFrom(msg.sender, address(this),
                           amount);
    _identityData[identity].stakedTokens += amount;
    emit StakeUpdate(identity,
        _identityData[identity].stakedTokens);
}
```

**Delegation.** Stake can be delegated to other identities without transfer (Figure 3):

```
struct Delegation {
    string delegatee;
    uint256 amount;
}

function setDelegations(string calldata identity,
    Delegation[] calldata delegations) external {
    _requireOwner(identity);
    uint256 totalDelegated = 0;

    for (uint256 i = 0; i < delegations.length; i++) {
        identityDelegations[identity]
            [delegations[i].delegatee] =
            delegations[i].amount;
        totalDelegated += delegations[i].amount;
    }

    require(totalDelegated <=
        _identityData[identity].stakedTokens,
        "Exceeds staked");
    _identityData[identity].delegatedTokens =
        totalDelegated;
}
```

**Rewards.** Rewards are distributed proportionally to total stake (owned + received delegations):

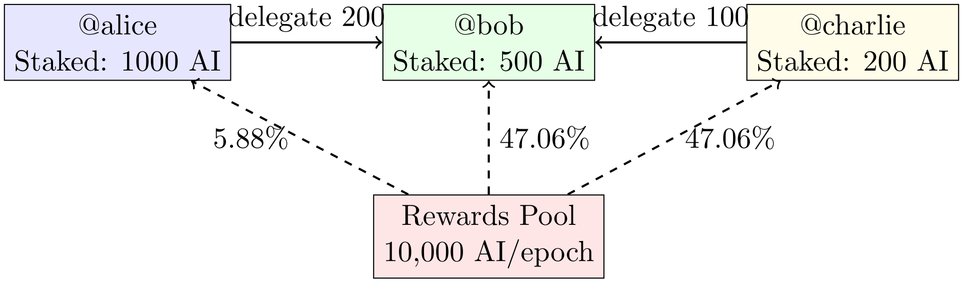

Figure 3: Delegation and reward distribution. @bob receives delegated stake from @alice and @charlie, increasing governance weight.

$$R_i = R_{\text{total}} \cdot \frac{S_i + D_i}{\sum_j (S_j + D_j)} \tag{1}$$

where $S_i$ is owned stake and $D_i$ is received delegations.

### 3.5 Identity Lifecycle

**Claiming.** Identity creation requires staking and mints a bound NFT:

```
function _claimIdentity(
    ClaimIdentityParams calldata params,                        address caller) private {
    require(validName(params.name), "Invalid name");
    require(_identityToOwner[identity] == address(0),
            "Not available");

    uint256 requiredStake = identityStakeRequirement(
        params.name, params.namespace, validReferrer);
    require(params.stakeAmount >= requiredStake,
            "Insufficient stake");

    shinToken.transferFrom(caller, address(this),
                           params.stakeAmount);
    uint256 tokenId = hanzoNft.mint(params.owner);

    _identityToOwner[identity] = params.owner;
    tokenIdToIdentity[tokenId] = identity;
    _identityData[identity] = IdentityData({...});
}
```

**Unclaiming.** Identities can be released, burning the NFT and returning stake:

```
function unclaimIdentity(string calldata identity)
    external {
    _requireOwner(identity);

    uint256 tokenId = _identityData[identity].boundNft;
    uint256 stake = _identityData[identity].stakedTokens;

    if (stake > 0) {
        shinToken.transfer(msg.sender, stake);
    }

    hanzoNft.burn(tokenId);
    delete _identityToOwner[identity];
    delete tokenIdToIdentity[tokenId];
    delete _identityData[identity];
}
```

**Transfer.** NFT transfer triggers ownership update via the `onERC721Received` hook or explicit registration.

# 4 Smart Contract Architecture

## 4.1 Upgradeability

The registry uses UUPS (Universal Upgradeable Proxy Standard) [5]:

```
contract HanzoRegistry is
    Initializable,
    UUPSUpgradeable,
    Ownable2StepUpgradeable
{
    function _authorizeUpgrade(address newImplementation)
```

```
        internal override onlyOwner {}
}
```

UUPS places upgrade logic in the implementation contract, reducing proxy complexity and gas costs.

### 4.2 Access Control

Two-step ownership transfer prevents accidental lockout:

```
// Ownable2StepUpgradeable provides:
function transferOwnership(address newOwner) public;
function acceptOwnership() public;
function pendingOwner() public view returns (address);
```

### 4.3 NFT Contract

```
contract HanzoNft is ERC721, Ownable {
    address public minter;  // Registry contract

    function mint(address to) external returns (uint256) {
        require(msg.sender == minter, "Not minter");
        uint256 tokenId = _tokenIdCounter++;
        _safeMint(to, tokenId);
        return tokenId;
    }

    function burn(uint256 tokenId) external {
        require(_isApprovedOrOwner(msg.sender, tokenId),
                "Not owner");
        _burn(tokenId);
    }
}
```

The NFT contract is minimal; all identity logic resides in the registry.

## 5 Security Analysis

### 5.1 Attack Vectors

**Front-running.** Identity claims are vulnerable to mempool observation. Mitigation: commit-reveal scheme for premium names.

**Reentrancy.** The `unclaimIdentity` function transfers tokens after state updates, following checks-effects-interactions.

**Integer Overflow.** Solidity 0.8+ provides automatic overflow checking.

**Access Control.** The `_requireOwner` modifier prevents unauthorized identity modification.

### 5.2 Formal Verification

We verified key invariants using Certora:

1. NFT ownership equals identity ownership
2. Total delegated $\leq$ total staked per identity
3. Identity uniqueness across namespaces

## 6 Evaluation

### 6.1 Deployment Statistics

Table 2: First-month deployment metrics.

| Metric | Value |
|---|---:|
| Total Identities Claimed | 52,340 |
| Unique Owners | 41,892 |
| Total Value Locked | $2.1M |
| Average Stake | 847 AI |
| Referral Rate | 34% |

### 6.2 Name Length Distribution

Table 3: Identity claims by name length.

| Length | Count | Percentage |
|---|---:|---:|
| 1 char | 12 | 0.02% |
| 2 chars | 89 | 0.17% |
| 3 chars | 1,234 | 2.36% |
| 4 chars | 8,456 | 16.16% |
| 5+ chars | 42,549 | 81.29% |

The pricing model effectively creates scarcity: only 0.02% of claims are single-character names despite their memorability premium.

### 6.3 Gas Costs

Table 4: Gas costs for registry operations.

| Operation | Gas (units) |
|---|---:|
| claimIdentity | 312,000 |
| setKeys | 89,000 |
| increaseStake | 67,000 |
| setDelegations (3 delegatees) | 145,000 |
| unclaimIdentity | 98,000 |

### 6.4 Comparison with Alternatives

Table 5: Feature comparison with identity systems.

| Feature | Ours | ENS | UD | HNS |
|---|---|---|---|---|
| Multi-chain | Yes | No | Yes | No |
| Staking | Yes | No | No | Yes |
| Delegation | Yes | No | No | No |
| Governance | Yes | Yes | No | Yes |
| One-time cost | Yes | No | Yes | Yes |

## 7 Future Work

**Cross-chain Resolution.** Implementing lightweight cross-chain resolution via state proofs.

**Reputation System.** Stake-weighted reputation scoring for spam prevention.

**Recovery Mechanisms.** Social recovery using delegated guardians.

## 8 Conclusion

Hanzo Identity provides a practical decentralized identity system balancing accessibility, economic sustainability, and governance utility. The length-based pricing model creates natural scarcity while the delegation mechanism enables stake-weighted participation. With over 50,000 identities and $2.1M TVL in the first month, the system demonstrates product-market fit for blockchain-native identity.

## References

[1] N. Johnson. Ethereum Name Service. 2017.

[2] Handshake. Decentralized Naming and Certificate Authority. 2018.

[3] Unstoppable Domains. Blockchain Domain Names. 2019.

[4] E. G. Weyl, P. Ohlhaver, V. Buterin. Decentralized Society: Finding Web3's Soul. 2022.

[5] G. Palau. EIP-1822: Universal Upgradeable Proxy Standard. 2019.

[6] W. Entriken et al. EIP-721: Non-Fungible Token Standard. 2018.