zDoge Threat Model

Version: 2.1
Last Updated: January 2025 (Updated for V4 deployment)
Status: Final - Aligned with Security Audit and V4 Security Fixes

Executive Summary

This document defines what zDoge protects against, what it does not protect against, and the assumptions required for privacy. Understanding these limitations is essential for using zDoge safely and effectively.

Key Takeaway: zDoge provides **cryptographic privacy** for shielded transactions, but privacy also depends on **operational security** (OpSec) and **anonymity set size**.

What zDoge Protects Against

1. On-Chain Transaction Linkage

What's Protected: - **Shield (t→z)**: Deposit amount and sender address are hidden - **Transfer (z→z)**: Sender, recipient, and amount are completely hidden - **Swap (z→z)**: Token types and amounts are hidden - **Unshield (z→t)**: Which note was spent is hidden (but amount/recipient are visible)

How: - Zero-knowledge proofs prove ownership without revealing note details - Merkle tree provides anonymity set (1 in N correlation) - Nullifiers prevent double-spending without revealing which note was spent - Commitments hide amounts, tokens, and ownership

Example:

```
Alice shields 100 DOGE → Creates commitment C1
Bob shields 50 DOGE → Creates commitment C2
Alice transfers 30 DOGE to Charlie → Spends C1, creates C3 (change) and C4 (recipient)

On-chain sees:
- Shield events: C1, C2 (no amounts, no senders)
- Transfer event: nullifierHash, C3, C4 (no amounts, no sender/recipient)

Observer cannot link:
- Which commitment belongs to which user
- Which shield led to which transfer
- Who sent to whom
```

2. Balance Visibility

What's Protected: - Total shielded balance per user is hidden - Individual note amounts are hidden - Token types in shielded pool are hidden (for transfers/swaps)

How: - Notes are stored locally (not on-chain) - Only commitments (hashes) are on-chain - Commitments reveal nothing about amounts or ownership

Example:

```
Alice has 3 shielded notes: 10 DOGE, 50 DOGE, 100 DOGE
On-chain: Only sees 3 commitments (C1, C2, C3)
No one can determine:
- How much Alice has
- What tokens Alice has
- How many notes Alice has
```

3. Transaction Graph Analysis

What's Protected: - Cannot build transaction graph between shielded addresses - Cannot trace funds through shielded pool - Cannot determine transaction flow

How: - Each transfer creates new commitments - No link between input and output commitments - Nullifiers don't reveal which commitment was spent

Example:

```
Alice → Bob → Charlie (all shielded)

On-chain sees:
- Transfer 1: nullifierHash1, C2, C3
- Transfer 2: nullifierHash2, C4, C5

Observer cannot determine:
- That Transfer 2 used C2 or C3 from Transfer 1
- That Bob was the intermediate recipient
- The transaction chain
```

4. Double-Spending

What's Protected: - Cannot spend the same note twice - Cannot create fake notes - Cannot mint tokens

How: - Nullifier hashes prevent double-spending - Merkle tree membership proofs prevent fake notes - Value conservation in circuits prevents minting

Example:

```
Alice tries to spend same note twice:
```

```
- First spend: nullifierHash1 marked as spent
- Second spend: nullifierHash1 already spent   (rejected)
```

What zDoge Does NOT Protect Against

1. Unshield Visibility

What's Visible: - **Recipient address** (public Ethereum address) - **Amount** (how much was withdrawn) - **Token type** (which token was withdrawn) - **Timestamp** (when withdrawal happened) - **Relayer address** (if using relayer)

Why: - Unshield must transfer tokens to a public address - Public addresses are visible on-chain by design - Amount must be visible for token transfer

Mitigation: - Use **fresh addresses** for each unshield - **Vary amounts** (don't unshield exact round numbers) - **Wait between transactions** (don't unshield immediately after shield) - **Use different addresses** for shield vs unshield

Example:

```
 BAD:
1. Shield 100 DOGE from address 0xAlice
2. Immediately unshield 100 DOGE to 0xAlice
→ Easy to correlate (same address, same amount, same time)

 GOOD:
1. Shield 100 DOGE from address 0xAlice
2. Wait 1 week, let anonymity set grow
3. Unshield 95.234 DOGE to fresh address 0xBob (never used before)
→ Hard to correlate
```

2. Timing Correlation

What's Vulnerable: - Shield → Transfer timing patterns - Transfer → Unshield timing patterns - Regular transaction schedules

Why: - If you shield at 2:00 PM and transfer at 2:05 PM, correlation is possible - Small anonymity set + unique timing = high correlation risk

Mitigation: - **Vary timing randomly** (don't create patterns) - **Wait between transactions** (let anonymity set grow) - **Use different times of day** (avoid predictable schedules)

Example:

```
 BAD:
- Always shield on Monday 9 AM
- Always transfer on Tuesday 10 AM
- Always unshield on Wednesday 11 AM
```

```
→ Pattern is obvious

  GOOD:
- Shield: Random times, random days
- Transfer: Wait 1-7 days, random times
- Unshield: Wait weeks/months, random times
```

3. Amount Correlation

What's Vulnerable: - Round number amounts (100, 1000, etc.) - Unique amounts (123.456789 DOGE) - Amount patterns across transactions

Why: - Round numbers are suspicious (most people use them) - Unique amounts can be traced if used multiple times - Amount patterns can link transactions

Mitigation: - **Vary amounts slightly** (100.123 DOGE instead of 100 DOGE) - **Don't reuse unique amounts** - **Use different amounts** for shield vs unshield

Example:

```
  BAD:
- Shield: 100 DOGE
- Unshield: 100 DOGE
→ Easy correlation

  GOOD:
- Shield: 100.234 DOGE
- Unshield: 98.567 DOGE (after fees)
→ Harder correlation
```

4. Network-Level Metadata

What's Vulnerable: - **IP address** (can be logged by RPC providers, frontend servers) - **Browser fingerprinting** (device, browser, extensions) - **RPC endpoint logs** (which RPC you use, when you use it) - **Frontend analytics** (if enabled)

Why: - zDoge runs in browser (needs network access) - RPC providers may log requests - Frontend servers may log access

Mitigation: - **Use VPN or Tor** (hide IP address) - **Use privacy-focused browser** (Firefox with privacy extensions) - **Use privacy-focused RPC** (or run your own) - **Disable analytics** (if frontend has them) - **Use incognito mode** (reduces fingerprinting)

Example:

```
  BAD:
```

```
- Connect from home IP
- Use default RPC endpoint
- Use Chrome with tracking extensions
→ IP, RPC, browser all logged

  GOOD:
- Use Tor or VPN
- Use privacy-focused RPC (or self-hosted)
- Use Firefox with uBlock Origin
→ Much harder to track
```

5. Frontend Compromise

What's Vulnerable: - Malicious frontend code injection (CDN compromise, DNS hijacking, hosting provider breach) - Browser extensions with excessive permissions - Phishing websites (fake zDoge sites) - Subresource integrity failures - Content Security Policy bypasses

Why: - Frontend runs in user's browser with access to wallet - Compromised frontend can exfiltrate spending keys during proof generation - Malicious code can intercept transaction data - Browser extensions can access localStorage and page content

Impact: - Complete loss of privacy (all notes decryptable) - Complete loss of funds (spending keys compromised) - No recovery mechanism once keys are exposed

Mitigation: - Verify website URL (check for typosquatting) - Use official source (GitHub, verified domain with SSL) - Review code changes (if technical, monitor GitHub) - Don't install untrusted browser extensions - Use hardware wallet integration (when available) - Implement Subresource Integrity (SRI) for all scripts - Implement strict Content Security Policy (CSP) - Consider IPFS hosting with hash verification - Use browser extension for frontend integrity verification (when available)

Current Status: - Frontend integrity verification not yet implemented - Critical for mainnet deployment - Recommendation: Implement SRI and CSP before mainnet launch

Example:

```
  BAD:
- Visit zdoge-cash.com (typo, phishing site)
- Install "zDoge helper" extension (malicious, steals keys)
- CDN compromised, malicious JavaScript injected
→ Complete compromise of privacy and funds

  GOOD:
- Visit zdoge.cash (official domain, verified SSL)
```

```
- No suspicious extensions installed
- Verify GitHub repository matches deployed code
- Browser extension verifies frontend hash
→ Much safer, but still requires trust in hosting
```

What's Vulnerable: - Malicious frontend code (if compromised) - Browser extensions (malicious extensions) - Phishing websites (fake zDoge sites)

Why: - Frontend runs in your browser - Has access to your wallet (if connected) - Can see your transactions

Mitigation: - **Verify website URL** (check for typos) - **Use official source** (GitHub, verified domain) - **Review code** (if technical, check GitHub) - **Don't trust browser extensions** (unless verified) - **Use hardware wallet** (additional security layer)

Example:

```
 BAD:
- Visit zdoge-cash.com (typo, phishing site)
- Install "zDoge helper" extension (malicious)
→ Funds at risk

 GOOD:
- Visit zdoge.cash (official domain)
- No suspicious extensions
- Verify GitHub repository
→ Much safer
```

6. Backend Indexer Dependency

What's Vulnerable: - Single backend indexer maintains Merkle tree state - Backend compromise could provide incorrect Merkle paths - Backend downtime prevents proof generation - Backend could censor specific users

Why: - Frontend requires Merkle tree state to generate proofs - Current implementation relies on centralized backend service - No fallback mechanism for Merkle path queries

Impact: - Cannot generate proofs if backend is unavailable - Incorrect Merkle paths could cause proof failures - Potential censorship if backend is compromised - Single point of failure for system functionality

Mitigation: - Implement decentralized indexer (future enhancement) - Allow direct RPC Merkle tree queries as fallback - Implement multiple indexer endpoints with failover - Cache Merkle paths locally when possible - Document backend dependency clearly to users

Current Status: - Acceptable for testnet deployment - Documented risk in

audit report - Recommendation: Implement fallback mechanisms before mainnet

7. Small Anonymity Set

What's Vulnerable: - Early users (small pool) - Rare token amounts - Unique transaction patterns

Why: - If only 10 notes exist, correlation is 1 in 10 - If only 1 note of 1000 DOGE exists, easy to identify - Small set = easier correlation

Mitigation: - **Wait for adoption** (let anonymity set grow) - **Use common amounts** (don't use unique amounts) - **Don't be first user** (if privacy-critical) - **Consider fixed-amount pools** (if added later)

Example:

```
 BAD:
- Shield 1,000,000 DOGE when pool is empty
- Only note in pool
→ 100% correlation (you're the only one)

 GOOD:
- Wait until pool has 1000+ notes
- Shield common amounts (10, 100, 1000 DOGE)
→ 1 in 1000+ correlation
```

8. User Error

What's Vulnerable: - **Losing spending key** → Permanent loss of funds - **Sharing shielded address** → Links identity - **Linking transactions** → Self-doxxing - **Reusing addresses** → Correlation

Why: - zDoge is non-custodial (you control keys) - No recovery mechanism (by design) - User mistakes can't be fixed

Mitigation: - **Backup spending key securely** (offline, multiple locations) - **Never share shielded address** (keep it private) - **Don't link transactions** (don't post about them) - **Use fresh addresses** (for unshield)

Example:

```
 BAD:
- Store spending key in browser (can be lost)
- Post shielded address on Twitter
- Unshield to same address you use for public transactions
→ Privacy compromised

 GOOD:
- Store spending key in hardware wallet + encrypted backup
```

```
- Never share shielded address
- Unshield to fresh addresses only
→ Privacy maintained
```

9. Relayer Censorship

What's Vulnerable: - Single relayer can censor transactions - Relayer can see transaction timing - Relayer can refuse to submit transactions

Why: - Current implementation uses single relayer - Relayer must see transaction to submit it - No alternative relayers yet

Mitigation: - **Submit directly** (pay gas yourself, bypass relayer) - **Use multiple relayers** (when available) - **Wait for decentralization** (roadmap item)

Example:

```
 BAD:
- Rely only on relayer
- Relayer goes down or censors you
→ Transactions fail

 GOOD:
- Have DOGE for gas (can submit directly)
- Use relayer when convenient
- Don't depend on relayer for critical transactions
```

Privacy Guarantees by Operation

Shield (t→z)

Hidden: - Sender address (if using relayer or different address) - Amount (commitment hides it) - Token type (commitment hides it)

Visible: - Shield event (commitment, leafIndex, timestamp) - Deposit transaction (if not using relayer, sender visible)

Privacy Level: High (if using relayer or different address)

Transfer (z→z)

Hidden: - Sender address - Recipient address - Amount - Token type - Transaction link

Visible: - Transfer event (nullifierHash, outputCommitments, timestamp) - Relayer address (if using relayer)

Privacy Level: Maximum (best privacy operation)

Swap (z→z)

Hidden: - Input token type - Output token type - Input amount - Output amount - Exchange rate

Visible: - Swap event (nullifierHash, outputCommitment, timestamp)

Privacy Level: Maximum (same as transfer)

Unshield (z→t)

Hidden: - Which note was spent - Sender address - Shielded address

Visible: - Recipient address (public address) - Amount (withdrawal amount) - Token type - Timestamp - Relayer address (if using relayer)

Privacy Level: Medium (lowest privacy operation)

Anonymity Set Model

How Anonymity Works

Anonymity Set = Number of shielded notes in the pool

```
Small Set (10 notes):    1 in 10 chance of correlation (10% risk)
Medium Set (100 notes):  1 in 100 chance (1% risk)
Large Set (1000 notes):  1 in 1000 chance (0.1% risk)
Very Large (10k notes):  1 in 10,000 chance (0.01% risk)
```

Factors Affecting Anonymity

1. **Pool Size**: More notes = better privacy
2. **Token Type**: Popular tokens = larger sets
3. **Amount Distribution**: Common amounts = better privacy
4. **Time**: Older notes = harder to correlate

Current Limitations

- **Testnet**: Small user base = small anonymity set
- **Variable Amounts**: Reduces uniformity (but increases flexibility)
- **Early Adoption**: First users have lower privacy

Future Improvements

- **Fixed-Amount Pools**: Optional pools for maximum anonymity
- **Anonymity Metrics**: Dashboard showing set size
- **Privacy Recommendations**: UI suggestions based on set size

---

Attack Scenarios

Scenario 1: Timing Correlation Attack

Attack: 1. Attacker monitors shield events 2. Attacker monitors transfer events 3. Attacker correlates by timing (shield at 2:00 PM, transfer at 2:05 PM)

Success Rate: - Small anonymity set: High (50%+) - Large anonymity set: Low (<1%)

Mitigation: - Wait between transactions - Vary timing randomly - Use larger anonymity set

Scenario 2: Amount Correlation Attack

Attack: 1. Attacker knows you shielded 100 DOGE 2. Attacker sees unshield of 100 DOGE 3. Attacker correlates by amount

Success Rate: - Unique amount: High (if only one) - Common amount: Low (if many)

Mitigation: - Vary amounts slightly - Don't use round numbers - Use different amounts for shield vs unshield

Scenario 3: Network Metadata Attack

Attack: 1. Attacker controls RPC provider 2. Attacker logs IP addresses 3. Attacker correlates by IP + timing

Success Rate: - Without VPN: High (if RPC logs) - With VPN/Tor: Low (IP hidden)

Mitigation: - Use VPN or Tor - Use privacy-focused RPC - Self-host RPC if possible

Scenario 4: Frontend Compromise Attack

Attack: 1. Attacker compromises CDN, DNS, or hosting provider 2. Attacker injects malicious JavaScript into frontend 3. Malicious code intercepts spending keys during proof generation 4. Attacker exfiltrates keys to remote server 5. Attacker can decrypt all user notes and spend funds

Success Rate: - If user visits compromised site: High (100% if keys in memory) - If frontend integrity verified: Low (0% if verification works) - If user uses hardware wallet: Low (keys never in browser)

Mitigation: - Verify website URL and SSL certificate - Check GitHub for official code and verify hash - Use hardware wallet (keys never leave device) - Don't install untrusted browser extensions - Implement Subresource Integrity (SRI) for all scripts - Implement Content Security Policy (CSP) - Consider IPFS

hosting with hash verification - Use browser extension for frontend integrity checks

Critical for Mainnet: Frontend integrity verification must be implemented before mainnet launch. This is a critical security requirement.

Scenario 5: Network Metadata Attack

Attack: 1. Attacker controls RPC endpoint or monitors network traffic 2. Attacker logs IP addresses of users querying specific commitments 3. Attacker correlates: IP address → commitments → user identity 4. Attacker tracks user activity patterns over time

Success Rate: - Without VPN/Tor: High (if RPC logs IP addresses) - With VPN: Medium (VPN provider could log) - With Tor: Low (IP address hidden)

Mitigation: - Use VPN or Tor when accessing zDoge.cash - Rotate RPC endpoints frequently - Consider running own RPC node - Use privacy-focused RPC providers - Implement RPC endpoint rotation in frontend

Critical for Privacy: Network metadata leakage is a high-risk privacy threat. All users should use VPN or Tor, especially high-value users.

Scenario 6: Small Anonymity Set Attack

Attack: 1. Attacker knows pool is small (10 notes) 2. Attacker knows you're one of the users 3. Attacker correlates all transactions

Success Rate: - Small set: High (10-50%) - Large set: Low (<1%)

Mitigation: - Wait for adoption - Use common amounts - Don't be first user (if privacy-critical)

Assumptions and Trust Model

What We Trust

1. **Smart Contracts**: Immutable, verified code
2. **Zero-Knowledge Proofs**: Mathematically sound
3. **Merkle Tree**: Correct implementation
4. **Nullifiers**: Prevent double-spending

What We Don't Trust (Minimal Trust)

1. **Relayer**: Can censor transactions, but cannot steal funds (proofs verify on-chain)
2. **Indexer**: Can degrade UX or provide incorrect Merkle paths, but cannot spend funds

3. **RPC Providers**: Can log metadata (IP addresses, queries), but cannot break cryptographic privacy
4. **Frontend**: Can be compromised (critical risk), keys are local but vulnerable during proof generation
5. **CDN/Hosting**: Can inject malicious code, must verify frontend integrity
6. **Browser Extensions**: Can access localStorage and page content, high risk if malicious

Trust Assumptions

For Maximum Privacy: - At least one honest participant in trusted setup ceremony - Smart contracts are correctly implemented (needs audit) - Circuits are correctly implemented (needs audit) - Users follow OpSec best practices

For Basic Privacy: - Smart contracts work as designed - Users don't make obvious mistakes - Anonymity set is reasonable (>100 notes)

Best Practices Summary

Do

- **Wait between transactions** (let anonymity set grow)
- **Vary amounts and timing** (don't create patterns)
- **Use fresh addresses** (for unshield)
- **Backup spending key securely** (offline, encrypted)
- **Use VPN or Tor** (hide IP address)
- **Verify website URL** (avoid phishing)
- **Use privacy-focused browser** (Firefox with extensions)

Don't

- **Don't shield and immediately transfer** (timing correlation)
- **Don't use round numbers** (amount correlation)
- **Don't unshield to same address** (address correlation)
- **Don't share shielded address** (identity correlation)
- **Don't link transactions publicly** (self-doxxing)
- **Don't rely only on relayer** (have gas for direct submission)
- **Don't use unique amounts** (if privacy-critical)

Privacy Levels

Level 1: Basic Privacy (Default)

Protection: - On-chain transaction linkage hidden - Balance hidden - Basic anonymity set

Requirements: - Use zDoge normally - Don't create obvious patterns

Use Case: General privacy needs

Level 2: Enhanced Privacy

Protection: - Level 1 + - Timing correlation reduced - Amount correlation reduced

Requirements: - Wait between transactions - Vary amounts and timing - Use fresh addresses

Use Case: Higher privacy needs

Level 3: Maximum Privacy

Protection: - Level 2 + - Network metadata hidden - Maximum anonymity set

Requirements: - Use VPN or Tor - Wait for large anonymity set - Follow all best practices - Use privacy-focused tools

Use Case: Critical privacy needs

Limitations and Disclaimers

What zDoge Cannot Do

1. **Cannot protect against user error** (losing keys, sharing addresses)
2. **Cannot protect against frontend compromise** (verify website)
3. **Cannot protect against network metadata** (use VPN/Tor)
4. **Cannot protect against small anonymity set** (wait for adoption)
5. **Cannot protect against timing correlation** (if patterns are obvious)
6. **Cannot protect against regulatory action** (compliance is user's responsibility)

What zDoge Assumes

1. **Users understand the system** (read documentation)
2. **Users follow best practices** (OpSec)
3. **Anonymity set is reasonable** (>100 notes for good privacy)
4. **Smart contracts are correct** (needs audit)
5. **Circuits are correct** (needs audit)
6. **Trusted setup is secure** (at least one honest participant)

Regulatory Disclaimer

zDoge is a **tool**, not a service. Users are responsible for: - Understanding local regulations - Complying with applicable laws - Paying taxes (if required) - Using the protocol legally

zDoge does not: - Provide legal advice - Guarantee anonymity - Protect against regulatory action - Endorse illegal use

---

Conclusion

zDoge provides **strong cryptographic privacy** for shielded transactions, but privacy is a **combination of cryptography and operational security**.

Key Points: - On-chain privacy is strong (ZK proofs, Merkle tree) - Off-chain privacy requires user effort (OpSec) - Small anonymity set reduces privacy - User errors can compromise privacy

For Maximum Privacy: 1. Use zDoge correctly (follow best practices) 2. Wait for adoption (larger anonymity set) 3. Use additional tools (VPN, Tor, privacy browser) 4. Understand limitations (this document)

Privacy is a practice, not just a tool.

Security Audit Alignment

This threat model has been reviewed and updated to align with the comprehensive security audit conducted in January 2025. Key updates include:

- Enhanced frontend compromise analysis with specific attack vectors
- Added backend indexer dependency as documented threat
- Expanded network metadata attack scenarios
- Updated mitigation strategies based on audit findings
- Aligned with Privacy Review document recommendations

Audit Reference: See `AUDIT_REPORT.md` for comprehensive security analysis and `PRIVACY_REVIEW.md` for detailed privacy threat analysis.

Document Maintenance

This threat model will be updated as: - New attack vectors are discovered - New mitigations are implemented - System architecture changes - Security audit findings are incorporated - Privacy review findings are incorporated

Last Review: January 2025 (aligned with security audit and V4 deployment)
Next Review: Quarterly or after major security/privacy updates

Contributors: zDoge Development Team, Security Audit Team

Feedback: Please report issues or suggestions via GitHub Issues