KISS Chain

Keep It Super Simple

The Most Mainstreamest Token on the Planet

Technical Whitepaper

Version 0.2 (Draft) • February 2026

KISS Chain — Technical Whitepaper

KISS Coin — Technical Specification & Whitepaper Outline

> ***Keep It Super Simple*** *Version: 0.2 (Draft) Date: 2026-02-16 Status: Active Development — Phase 2 (Consensus & Multi-Node) Changes: Added* `intl_id` *protocol-native compliance identity field*

---

1. Abstract

KISS Coin is a cryptocurrency and blockchain designed around a single principle: **radical simplicity**. Every design decision — from consensus to APIs to wallet formats — is made to minimize complexity while maintaining security and

decentralization.

The crypto industry has a complexity problem. Smart contract platforms require months of developer onboarding. Users need tutorials to send tokens. Wallets have dozens of settings. KISS takes the opposite approach: **if it can’t be explained in one sentence, it doesn’t belong.**

Ticker: `KISS` **Total Supply:** 100,000,000,000,000 (100 trillion, fixed) **Decimals:** 3 (0.001 KISS smallest unit — simple, human-readable) **Block Time:** 10 seconds **Transaction Fee:** Zero — sending KISS is free (governance-adjustable framework retained)

---

2. Design Philosophy

The KISS Rules

Every protocol decision must pass these filters:

1. **One way to do it** — No alternative methods, no configuration options
2. **Explain it to a teenager** — If the concept needs a PhD, simplify it
3. **curl is your SDK** — The API must be usable with basic HTTP tools
4. **Read the address out loud** — Identifiers must be human-friendly
5. **No surprises** — No fees, confirmations are final, behavior is predictable

What KISS Intentionally Does NOT Do

Feature	Why Not
Smart contracts / VM	Complexity explosion — use webhooks + off-chain logic instead
Variable gas fees	Unpredictable costs confuse users — KISS has no fee at all
Multiple token standards	One standard, one interface, no exceptions
Sharding / L2 / rollups	Scaling complexity — optimize the base layer instead
Governance tokens / DAOs	Keep governance simple — rough consensus + running code
NFTs (at launch)	Scope creep — add later only if it stays simple

3. Technical Architecture

3.1 Cryptography — One Algorithm Per Purpose

Purpose	Algorithm	Why
Hashing	BLAKE3	Fastest secure hash, single output size (256-bit)
Signatures	Ed25519	Simple, fast, no configuration (unlike ECDSA curve options)
Key derivation	BLAKE3-KDF	Same family as hashing — one less dependency
Address format	`kiss1` + Bech32m	Human-readable, checksum-protected, one format

Example address: `kiss1qpzry9x8gf2tvdw0s3jn54khce6mua7l`

No hex addresses. No `0x` prefix confusion. No checksum-case-sensitivity (looking at you, EIP-55).

3.2 Consensus — Simple Proof-of-Stake

Property	Value
Mechanism	Delegated Proof-of-Stake (simplified)
Validators	Max 100 active validators
Block production	Round-robin among top 100 by stake
Finality	Instant (single block) — no probabilistic confirmations
Slashing	Simple: miss 100 blocks → removed from active set. No complex penalty math
Minimum stake	1,000,000,000 KISS (1 billion)
Delegation	One command: `kiss delegate <validator> <amount>`
Reward	5% annual inflation, distributed proportionally to stakers

No committees. No epochs. No randomness beacons. You stake, you validate in order, you earn.

3.3 Transaction Model

One Transaction Type

Every transaction has the same structure:

```
(#cb1-1){
(#cb1-2)  "from": "kiss1abc...",
(#cb1-3)  "to": "kiss1def...",
(#cb1-4)  "amount": "5000.000",
(#cb1-5)  "memo": "Payment for coffee",
(#cb1-6)  "fee": "0.000",
(#cb1-7)  "intl_id": ""
(#cb1-8)}
```

That's it. No `type` field. No `data` field. No `nonce` — the chain handles ordering. No `gasLimit` — there is no fee.

The `intl_id` field is an optional compliance identity reference (see Section 3.6 — INTL.ID). When set, it signals that the sender claims a verified KYC/AML identity on the INTL.ID standard. When empty, the transaction is fully anonymous — identical behavior to any other blockchain.

Transaction Lifecycle

```
Created → Signed → Submitted → Confirmed (1 block, final)
```

No "pending" limbo. No "dropped" transactions. No "replace-by-fee". Submit → confirmed or rejected. Done.

Fees

- **Zero fees** — sending KISS is free. The amount sent equals the amount received. Always.
- **No fee market** — first-in, first-out ordering
- **Framework retained** — the `fee` field exists in every transaction and `ConsensusParams.FeeAmount` is governance-adjustable. If the community ever decides fees are needed, the protocol requires no structural change — validators simply accept the new consensus fee value via software upgrade
- **Why zero?** Spam defense is provided by rate limits + minimum balance, not economic fee cost. At low market caps, fees are economically meaningless

anyway (1,000 KISS at $1M market cap = $0.00001). Rate limits carry the defense at every market cap stage, making fees redundant.

Send Rate Limits (Balance-Based)

Sending transactions is rate-limited based on the sender's current balance. Receiving is always unlimited. This prevents spam at all market cap levels, even when fees are economically insignificant.

Minimum balance to send: 100,000 KISS. Wallets below this threshold can receive but cannot send. This is not a fee — the capital stays in the wallet. It's a floor that prevents an attacker from creating millions of dust wallets to bypass rate limits.

Sender Balance	Max Send Rate	Transactions/Hour
< 100,000 KISS	**Cannot send**	0
≥ 100,000 KISS (100K)	1 tx / 5 min	12/hour
≥ 10,000,000 KISS (10M)	1 tx / min	60/hour
≥ 100,000,000 KISS (100M)	1 tx / 10 sec	360/hour
≥ 1,000,000,000 KISS (1B)	1 tx / sec	3,600/hour

- **Sending only** — merchants and exchanges can receive unlimited transactions per second
- **Balance checked at submission time** — the node already reads the sender's balance to verify sufficient funds; zero additional cost
- **Consensus-level rule** — validators reject blocks containing transactions that violate rate limits
- **Minimum balance caps wallet splitting** — an attacker with 1B KISS can create at most 10,000 wallets (1B / 100K), capping total spam at 120,000 tx/hr. Without the minimum, the same attacker could create 1M dust wallets for 12M tx/hr — 100× worse
- **Not a fee — capital stays in wallet** — the 100K minimum is a floor, not consumed. You can spend down to 100K and still send. Below 100K, you can only

receive.

- **Faucet must give ≥ 100K KISS** — so new users can send immediately after claiming
- **All thresholds are governance-adjustable** — see Section 3.7

3.4 Blocks

```
(#cb3-1){
(#cb3-2)  "height": 1,
(#cb3-3)  "hash": "blake3_abc123...",
(#cb3-4)  "previous_hash": "blake3_000000...",
(#cb3-5)  "validator": "kiss1val...",
(#cb3-6)  "timestamp": "2026-06-01T00:00:10Z",
(#cb3-7)  "transactions": [],
(#cb3-8)  "tx_count": 0
(#cb3-9)}
```

No uncles. No state roots. No receipt trees. One flat list of transactions per block.

3.5 Wallet Format

- **Default: one seed = one address** — simple, no choices
- **Available: HD derivation** — unlimited addresses from one seed, for power users and businesses
- **Backup:** 12-word mnemonic (BIP39 compatible for familiarity)
- **Creation:** One command or one API call

Default (Regular Users)

```
(#cb4-1)$ kiss wallet create
(#cb4-2)Address:  kiss1qpzry9x8gf2tvdw0s3jn54khce6mua7l
(#cb4-3)Mnemonic: apple banana cherry ... (12 words)
(#cb4-4)Done. That's your wallet.
```

HD Wallet (Power Users, Businesses, Exchanges)

One seed phrase generates unlimited addresses, organized into accounts:

```
Master Seed (12-word phrase)
  └── Master Key (Ed25519)
        ├── Account 0 (personal)
        │     ├── Address 0  kiss1aaa...
        │     ├── Address 1  kiss1bbb...
        │     └── Address 2  kiss1ccc...
        ├── Account 1 (business)
        │     ├── Address 0  kiss1ddd...
        │     └── Address 1  kiss1eee...
        └── Account N...
```

```
(#cb6-1)# Derive additional addresses from same seed
(#cb6-2)$ kiss wallet create --account 1
(#cb6-3)Address:  kiss1xy7r8q5gf3tvdw0s4kn65lhdf7nub8m
(#cb6-4)Account:  1
(#cb6-5)
(#cb6-6)$ kiss wallet create --account 2
(#cb6-7)Address:  kiss1mn9p4z3hg6svew1t5jo76midg8ovc9n
(#cb6-8)Account:  2
(#cb6-9)
(#cb6-10)# List all derived addresses
(#cb6-11)$ kiss wallet list
(#cb6-12)Account 0:  kiss1qpzry9x8gf2tvdw0s3jn54khce6mua7l
(default)
(#cb6-13)Account 1:  kiss1xy7r8q5gf3tvdw0s4kn65lhdf7nub8m
(#cb6-14)Account 2:  kiss1mn9p4z3hg6svew1t5jo76midg8ovc9n
```

HD Derivation Path

```
m / purpose' / coin_type' / account' / address_index
m / 44'      / KISS'      / 0'       / 0
```

Following the SLIP-44 standard. KISS will register an official coin type number.

Why HD Matters

Use Case	How HD Helps
Privacy	Fresh address per transaction — harder to trace your full balance
Business	Separate address per customer/invoice — automatic reconciliation
Exchange	Unique deposit address per user — essential for exchange operations
Organization	Personal, business, savings — all from one seed
Backup	One 12-word phrase backs up ALL addresses, forever

The KISS Principle Still Applies

HD complexity is **entirely in the wallet, not the chain.** The blockchain sees individual addresses — it doesn't know or care if they came from one seed or separate keys. Zero impact on protocol, nodes, or block size. Users who don't need HD never see it.

3.6 INTL.ID — Protocol-Native Compliance Identity

KISS is the first blockchain with a **protocol-native compliance identity field**. Every transaction includes an optional `intl_id` parameter that references a verified KYC/AML identity on the INTL.ID (https://intl.id) standard.

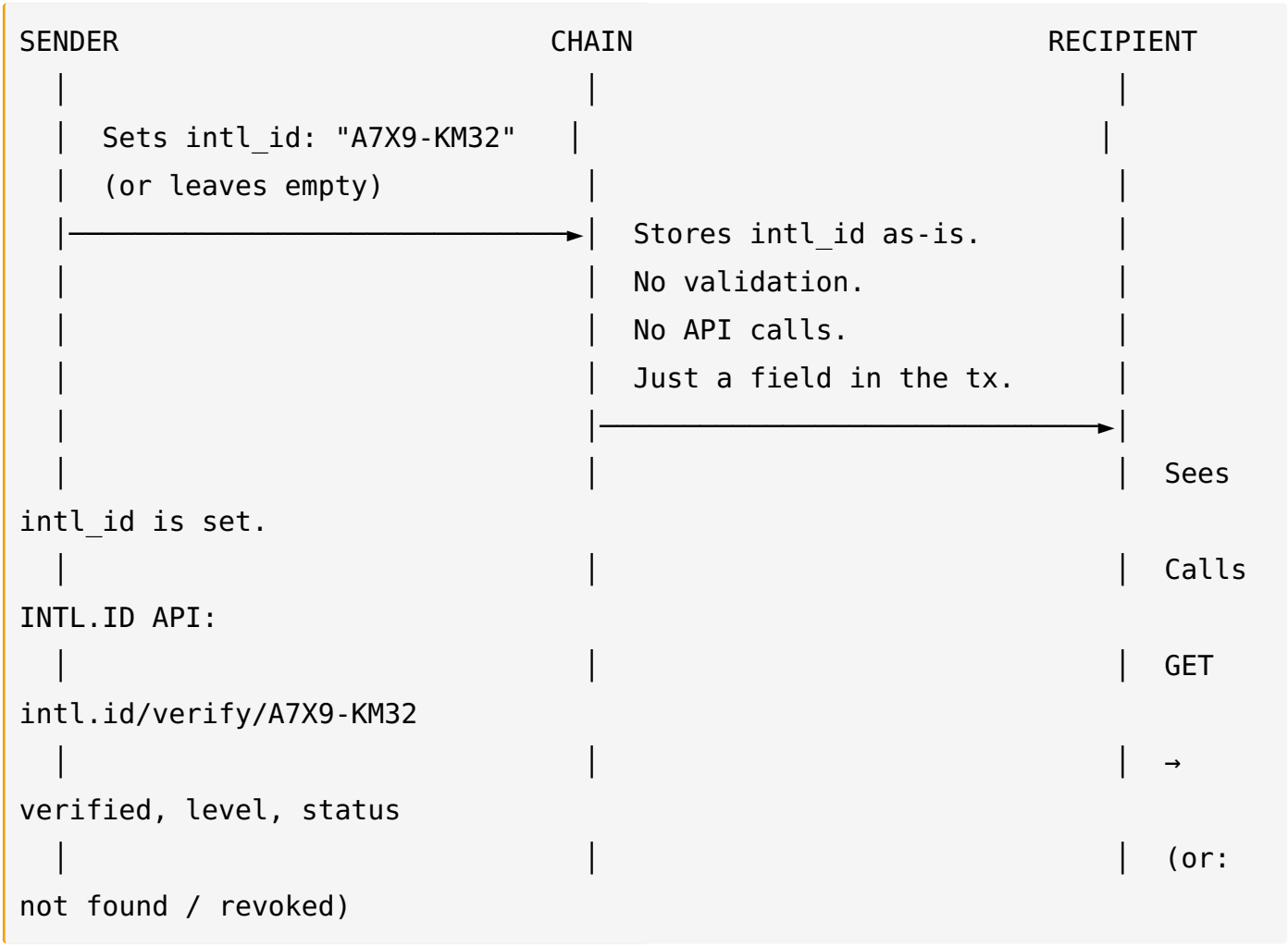
Why This Exists

Most blockchains treat compliance as an afterthought — bolted on by third-party analytics firms doing probabilistic chain analysis. KISS takes a different approach:

compliance is a first-class protocol feature, available to anyone who wants it, invisible to anyone who doesn't.

Approach	How it works	Problem
Ethereum / Bitcoin	No identity at protocol level. Chainalysis/Elliptic do forensic analysis after the fact.	Expensive, imprecise, requires specialized tooling
KISS	Optional `intl_id` field on every transaction. Recipient verifies off-chain via INTL.ID API.	None — opt-in, zero protocol overhead, standardized

How It Works



Design Principles

1. **The chain is dumb about `intl_id`.** It stores the value, includes it in the signed payload, and does nothing else. No validation, no API calls, no checksums. The chain records. The recipient verifies.

2. **Anyone can set any value.** The `intl_id` field accepts any string up to 128 bytes. There is no on-chain enforcement that the ID is real, valid, or belongs to the sender. This is intentional — the chain has no business authenticating identity. That is the recipient's responsibility.

3. **Presence is the signal.** A transaction with `intl_id` set says: "the sender claims a compliance identity exists." A transaction without it says nothing. Whether that claim is true is determined off-chain by anyone who cares to check.

4. **Verification is the recipient's job.** An exchange, merchant, or compliance officer who receives a KISS transaction with an `intl_id` can call the INTL.ID API to verify: is this ID real? What verification level? Is it still active? This is a standard REST API call — not a blockchain operation.

5. **No on-chain validation was deliberately rejected.** Having the chain call an external API per transaction would create a hard dependency (if INTL.ID is down, the chain halts), generate unsustainable bandwidth (millions of TPS × API calls), and violate decentralization principles. A checksum scheme was also rejected — trivially spoofable and adds complexity without real security.

Field Specification

Property	Value
Field name	`intl_id`
Type	String
Max length	128 bytes
Character set	ASCII printable (0x20–0x7E) — no Unicode, no control characters
Required	No — empty string or omitted = no compliance identity
Signed	Yes — included in the Ed25519 signed transaction payload
Validated by chain	No — stored as-is, no verification
Validated by recipient	Off-chain, via INTL.ID API (or any method the recipient chooses)

Why ASCII-Only, No Unicode

An identity field must be **unambiguous**. Unicode introduces: - **Homoglyph attacks** — Cyrillic “a” (U+0430) looks identical to Latin “a” (U+0061) - **Invisible characters** — zero-width joiners, right-to-left overrides - **Normalization issues** — NFC vs. NFD forms of the same character

ASCII printable characters eliminate all of these. 128 bytes of ASCII provides vastly more than enough combinations for any identifier scheme.

What Is INTL.ID?

INTL.ID (https://intl.id) is an independent KYC/AML identity standard and verification service. It defines:

- **A standardized KYC/AML process** — a set of identity verification checks

(government ID, proof of address, sanctions screening, PEP checks) that constitute a “verified” identity

- **A unique identity reference** — an opaque ID string that maps to a verified profile, without exposing personal data
- **A verification API** — a REST endpoint where any recipient can check the validity, status, and verification level of an INTL.ID

When a compliance officer, exchange, or merchant sees an `intl_id` on a KISS transaction, they know: - The sender claims to have completed the INTL.ID verification process - The ID can be checked against the INTL.ID API for authenticity and current status - The verification process follows a published, auditable standard - No personal data is exposed on-chain — only the reference ID

Example: Compliant Transaction

```
curl -X POST https://node.kisschain.org/v1/transaction \
  -d '{
    "from": "kiss1abc...",
    "to": "kiss1def...",
    "amount": "50000.000",
    "memo": "Invoice #4821",
    "intl_id": "A7X9-KM32-PQ81",
    "signature": "ed25519_sig..."
  }'
```

Example: Anonymous Transaction (Default)

```
curl -X POST https://node.kisschain.org/v1/transaction \
  -d '{
    "from": "kiss1abc...",
    "to": "kiss1def...",
    "amount": "50000.000",
    "memo": "Invoice #4821",
    "signature": "ed25519_sig..."
  }'
```

Both transactions are processed identically by the chain. Same speed. Same finality. No fee either way. The only difference is the compliance signal available to the recipient.

Use Cases

Scenario	How `intl_id` helps
Exchange deposit	Exchange sees `intl_id` on incoming tx → verifies against INTL.ID → streamlined compliance
Business payment	Merchant requires `intl_id` for orders above $10,000 → verifies sender identity off-chain
Cross-border transfer	FATF travel rule compliance — sender's identity is referenced, verifiable by receiving institution
Audit trail	Compliance officer queries historical transactions → `intl_id` provides identity linkage without on-chain PII
Voluntary compliance	Individual sets `intl_id` to demonstrate good faith — not required, but builds trust
Anonymous transaction	User omits `intl_id` → fully anonymous, no compliance signal, no difference in processing

What INTL.ID Is NOT

- **Not mandatory** — the protocol works identically with or without it
- **Not validated on-chain** — the chain does not call any external service
- **Not a deanonymization tool** — the ID is opaque; personal data is never on-chain
- **Not a single point of failure** — if INTL.ID is offline, transactions still process; only off-chain verification is affected
- **Not a government backdoor** — it’s opt-in, sender-controlled, and reveals nothing without an explicit API lookup by the recipient

3.7 Governance-Adjustable Parameters

Certain protocol parameters are designed to be adjusted over time as the network grows and KISS gains market value. These are **consensus parameters** — every node must agree on the current values.

Adjustable Parameters

Parameter	Launch Value	Why Adjustable
Transaction fee	0 KISS (zero)	Fee framework retained; governance can set a non-zero value if the community decides fees are needed
Minimum send balance	100,000 KISS	Floor to prevent dust-wallet spam; may need lowering as KISS appreciates
Send rate limit: default tier	1 tx / 5 min	May need tuning based on real-world usage patterns
Send rate limit: tier thresholds	100K / 10M / 100M / 1B KISS	Balance thresholds may shift as token economics evolve
Send rate limit: tier rates	1/5min, 1/min, 1/10sec, 1/sec	May need tuning based on network capacity
Minimum validator stake	1,000,000,000 KISS	May adjust as KISS value changes
Faucet amount	100,000 KISS	Must be ≥ minimum send balance so new users can transact

Fee Design Rationale

The fee is set to **zero by default** for the following reasons:

1. **Fees at low market caps are economically meaningless** — 1,000 KISS at a $1M market cap costs $0.00001. Spam prevention at early stages relies entirely on rate limits and minimum balance, not fees.
2. **Rate limits make fees redundant** — the balance-based send rate limits + minimum balance provide bounded spam protection at every market cap stage. There is no market cap level at which fees uniquely provide defense that rate limits do not.

3. **“Free KISS” is a powerful differentiator** — no major L1 blockchain offers zero-fee transactions. KISS now joins Nano as feeless, but with a richer feature set (tokens, staking, `intl_id`, gossip relay).
4. **Validators don’t depend on fees** — validator income comes entirely from block production rewards (5% annual inflation). Zero fee revenue is lost by setting fees to zero.
5. **Governance retains the option** — if circumstances change (rewards exhausted, spam attack at high market cap), the community can activate fees via software upgrade. The framework is live in every node; only the value is zero.

How Parameters Change (Open Question)

The mechanism for adjusting these parameters is an **open design question** with two candidate approaches:

Option A: Software Upgrade (Bitcoin Model)

All node operators upgrade to a new software version containing the new parameter values. This is how Bitcoin handles protocol changes.

Pros	Safest — no on-chain attack surface. Cannot be gamed. Requires broad community consensus.
Cons	Slow — requires coordinated upgrade across all nodes. Could take weeks/months.
Risk	Low — no mechanism to exploit. Worst case: disagreement leads to a fork (same as any blockchain).
Precedent	Bitcoin, Litecoin, most conservative chains.

Option B: Validator Voting (On-Chain Governance)

Validators vote on parameter changes by including proposed values in their block

headers. Supermajority (67%+) activates the change after a delay.

Pros	Faster — changes can activate in days. No software upgrade needed.
Cons	Validators who control 67%+ of stake can manipulate parameters.
Risk	Medium — a well-funded adversary who acquires sufficient stake could raise fees to make the network unusable, or adjust rate-limit thresholds. Economic self-interest (validators have staked 1B+ KISS each) provides some protection, but is not airtight.
Precedent	Cosmos, Tezos, Polkadot.

Current decision: Deferred. This will be resolved before mainnet launch. The conservative default is Option A (software upgrade). Option B may be considered if a sufficiently tamper-resistant design emerges.

What Is NOT Adjustable

These are **hardcoded protocol constants** that never change:

Constant	Value	Why Fixed
Total supply	100,000,000,000,000 KISS	Monetary policy must be immutable
Decimals	3	Changing decimals would break all existing balances
Block time	10 seconds	Fundamental to finality guarantees
Finality	1 block	Core protocol promise
Fee destination	Burned (destroyed)	If fees are ever activated via governance, they are burned — not paid to validators
Transaction format	Single type, flat JSON	Protocol simplicity is non-negotiable

---

4. API Design — One Page of Docs

Base URL

```
https://node.kisschain.org/v1
```

Every Endpoint

Method	Path	Description
`GET`	`/balance/{address}`	Get balance
`POST`	`/transaction`	Submit signed transaction
`GET`	`/transaction/{hash}`	Get transaction details + status
`GET`	`/address/{address}/history`	Transaction history (filterable)
`GET`	`/block/{height}`	Get block by height
`GET`	`/block/latest`	Get latest block
`GET`	`/status`	Node status
`GET`	`/validators`	Active validator list
`WebSocket`	`/stream`	Real-time transaction & block notifications

That’s the entire API. 8 REST endpoints + 1 WebSocket. No GraphQL schemas. No SDK required.

Example: Send KISS

```
# Check balance
curl https://node.kisschain.org/v1/balance/kiss1abc...

# Send
curl -X POST https://node.kisschain.org/v1/transaction \
  -H "Content-Type: application/json" \
  -d
'{"from":"kiss1abc...","to":"kiss1def...","amount":"5000.000","memo":"hi","in


# Confirm
curl
https://node.kisschain.org/v1/transaction/blake3_txhash...
```

Three commands. Done. No SDK required.

Response Format — Always the Same

```
{
  "ok": true,
  "data": { ... },
  "error": null
}
```

Every endpoint. Same structure. `ok` is a boolean. `data` has the payload. `error` is a string or null. No error codes to memorize — the message tells you what's wrong in plain English.

Transaction Tracking — Full Lifecycle Visibility

Every transaction on KISS is fully trackable from submission to confirmation. No ambiguity, no guessing.

Transaction Status Lifecycle

```
SUBMITTED → PENDING → CONFIRMED
                    → FAILED (with reason)
```

Only three possible states. No “dropped,” no “replaced,” no “stuck in mempool for 3 days.” A transaction is either confirmed in the next block (~10 seconds) or failed with a clear reason.

Transaction Response — Full Detail

```
curl https://node.kisschain.org/v1/transaction/blake3_abc123...
```

```
{
  "ok": true,
  "data": {
    "hash": "blake3_abc123...",
    "status": "confirmed",
    "from": "kiss1sender...",
    "to": "kiss1receiver...",
    "amount": "5000.000",
    "fee": "0.000",
    "memo": "Invoice #1234 — January rent",
    "intl_id": "A7X9-KM32-PQ81",
    "block_height": 84201,
    "block_hash": "blake3_block789...",
    "timestamp": "2027-01-15T14:32:10Z",
    "signature": "ed25519_sig...",
    "token": null,
    "explorer_url": "https://explorer.kisschain.org/transaction/blake3_abc123..."
  },
  "error": null
}
```

Every field a human or system needs — **in plain English, no hex encoding, no decoding required.**

Field	What it tells you
`hash`	Unique transaction ID — the receipt
`status`	confirmed, pending, or failed
`from` / `to`	Sender and receiver addresses
`amount`	How much was sent (human-readable, with decimals)
`fee`	Always 0 KISS (free)
`memo`	The sender’s note/reference/invoice number
`intl_id`	INTL.ID compliance identity reference (empty if not set)
`block_height`	Which block it’s in
`timestamp`	When it was confirmed (ISO 8601)
`token`	If this is a KISS-1 token transfer (e.g., “USDK”), otherwise null
`explorer_url`	Direct link to view on the block explorer

Failed Transaction Response

```
(#cb17-1){
(#cb17-2)  "ok": true,
(#cb17-3)  "data": {
(#cb17-4)    "hash": "blake3_def456...",
(#cb17-5)    "status": "failed",
(#cb17-6)    "reason": "Insufficient balance. Required: 5000.000
KISS. Available: 4500.000 KISS.",
(#cb17-7)    "from": "kiss1sender...",
(#cb17-8)    "to": "kiss1receiver...",
(#cb17-9)    "amount": "5000.000",
(#cb17-10)    "timestamp": "2027-01-15T14:32:10Z"
(#cb17-11)  },
(#cb17-12)  "error": null
(#cb17-13)}
```

Human-readable error messages. Not `0x08c379a0` — actual words explaining what went wrong.

Address History — Filterable

```
(#cb18-1)# Full history
(#cb18-2)curl
https://node.kisschain.org/v1/address/kiss1abc.../history
(#cb18-3)
(#cb18-4)# Filter by direction
(#cb18-5)curl
https://node.kisschain.org/v1/address/kiss1abc.../history?direction=sent
(#cb18-6)curl
https://node.kisschain.org/v1/address/kiss1abc.../history?
direction=received
(#cb18-7)
(#cb18-8)# Filter by date range
(#cb18-9)curl
https://node.kisschain.org/v1/address/kiss1abc.../history?from=2027-01-
01&to=2027-01-31
(#cb18-10)
(#cb18-11)# Filter by amount range
(#cb18-12)curl
https://node.kisschain.org/v1/address/kiss1abc.../history?
min_amount=1000&max_amount=50000
(#cb18-13)
(#cb18-14)# Filter by memo keyword
(#cb18-15)curl
https://node.kisschain.org/v1/address/kiss1abc.../history?memo=invoice
(#cb18-16)
(#cb18-17)# Filter by token (KISS-1 transfers only)
(#cb18-18)curl
https://node.kisschain.org/v1/address/kiss1abc.../history?token=USDK
(#cb18-19)
(#cb18-20)# Pagination (simple page-based, not cursor tokens)
(#cb18-21)curl
https://node.kisschain.org/v1/address/kiss1abc.../history?
page=1&per_page=50
```

```
{
  "ok": true,
  "data": {
    "address": "kiss1abc...",
    "total_transactions": 1247,
    "page": 1,
    "per_page": 50,
    "transactions": [
      {
        "hash": "blake3_abc123...",
        "direction": "sent",
        "to": "kiss1def...",
        "amount": "5000.000",
        "fee": "0.000",
        "memo": "Invoice #1234",
        "status": "confirmed",
        "block_height": 84201,
        "timestamp": "2027-01-15T14:32:10Z",
        "token": null
      },
      {
        "hash": "blake3_xyz789...",
        "direction": "received",
        "from": "kiss1ghi...",
        "amount": "250000.000",
        "memo": "Salary — January",
        "status": "confirmed",
        "block_height": 84195,
        "timestamp": "2027-01-15T14:31:00Z",
        "token": null
      }
    ]
  },
  "error": null
}
```

Simple page-based pagination. No opaque cursor tokens. `?page=2&per_page=50` — anyone can understand that.

Real-Time Notifications — WebSocket Stream

Connect once, receive live updates:

```
# Connect to WebSocket
wscat -c wss://node.kisschain.org/v1/stream
```

```
// Subscribe to an address
{"subscribe": "address", "address": "kiss1abc..."}

// Subscribe to all new blocks
{"subscribe": "blocks"}

// Subscribe to a specific pending transaction
{"subscribe": "transaction", "hash": "blake3_pending..."}
```

Events arrive as JSON:

```
// New transaction received at your address
{
  "event": "transaction_received",
  "data": {
    "hash": "blake3_new123...",
    "from": "kiss1sender...",
    "amount": "50000.000",
    "memo": "Payment for order #567",
    "status": "confirmed",
    "timestamp": "2027-01-15T14:33:20Z"
  }
}

// New block produced
{
  "event": "new_block",
  "data": {
    "height": 84202,
    "hash": "blake3_block...",
    "transaction_count": 147,
    "timestamp": "2027-01-15T14:33:20Z"
  }
}

// Pending transaction confirmed
{
  "event": "transaction_confirmed",
  "data": {
    "hash": "blake3_pending...",
    "block_height": 84202,
    "status": "confirmed"
  }
}
```

Use cases: - **Payment terminals:** “Ding! Payment received” in real-time - **Exchanges:** Instant deposit detection - **Wallets:** Live balance updates without polling - **Businesses:** Webhook-style integration via WebSocket

Block Explorer — explorer.kisschain.org

A single-page, minimal block explorer. No app to install, no wallet to connect — just a URL:

```
https://explorer.kisschain.org/transaction/blake3_abc123...
https://explorer.kisschain.org/address/kiss1abc...
https://explorer.kisschain.org/block/84201
```

Feature	What you see
Transaction page	Status, from, to, amount, memo, block, timestamp
Address page	Balance, full transaction history (filterable), staking info
Block page	Height, hash, timestamp, transaction list
Search	Paste any hash, address, or block number — auto-detects type
Token tracking	KISS-1 tokens (USDK, etc.) shown alongside native KISS

No account required. No ads. No “connect wallet” popups. Just data.

Tracking Comparison

Feature	Bitcoin	Ethereum	Solana	KISS
Transaction status	Unconfirmed → N confirmations (probabilistic)	Pending → confirmed (can reorg)	Processing → finalized	Pending → confirmed / failed (instant, final)
Error messages	None (just “failed”)	Hex-encoded revert reason	Program error codes	**Plain English explanation**
History filtering	Third-party only (Blockstream)	Third-party only (Etherscan)	Third-party only (Solscan)	**Built into every node’s API**
Real-time updates	ZMQ (complex setup)	WebSocket (requires Alchemy/Infura)	WebSocket (requires RPC provider)	**WebSocket on every node, free**
Memo/note field	OP_RETURN (80 bytes, hex)	Input data (hex-encoded)	Memo program (requires instruction)	**Plain text memo field, native**
Compliance identity	None — third-party analytics only	None — third-party analytics only	None	**Protocol-native `intl_id` field (opt-in)**
Block explorer	Third-party (blockchain.com)	Third-party (etherscan.io)	Third-party (solscan.io)	**Official, built-in, open-source**

---

5. Token Economics

Parameter	Value
Total supply at genesis	100,000,000,000,000 KISS (100 trillion)
Decimals	3 (0.001 KISS minimum unit)
Integer type	uint64 (native 64-bit, 8 bytes, 184× headroom)
Staking reward	5% annual inflation
Transaction fee	0 KISS (free — governance-adjustable framework retained)
Distribution	See below

Genesis Distribution

Allocation	%	KISS	Vesting / Release
Public sale	40%	40,000,000,000,000	None — immediately tradeable
Staking rewards pool	25%	25,000,000,000,000	Released over 10 years (block rewards)
Team & development	15%	15,000,000,000,000	4-year vesting schedule
Ecosystem grants	10%	10,000,000,000,000	Faucet, exchange listings, dev grants, partnerships
Reserve	10%	10,000,000,000,000	Locked 1 year, then governed by community

> ***Canonical source:*** `internal/chain/genesis.go` *and README-KISS-COIN-ECOSYSTEM-002.md (updated 2026-02-17).*

Strategic Reserve — 10%

The strategic reserve functions as a treasury for the long-term health of the KISS

ecosystem. It is **not** a personal fund — it is controlled by the KISS Foundation with full transparency.

Permitted Uses

Use Case	Description
Exchange listings	Provide liquidity for centralized exchange listings (Coinbase, Binance, etc.)
Market stabilization	Inject or withdraw liquidity during extreme price volatility
Strategic partnerships	Fund integrations with payment processors, wallets, and platforms
Emergency funding	Bridge funding if the team/development allocation is exhausted before self-sustainability

Release Rules

- **Locked at genesis** — no tokens released until the first milestone is met
- **Milestone-based unlocks** — tokens are released in tranches tied to verifiable milestones:
  - Mainnet launch: up to 2% (2T KISS)
  - First exchange listing: up to 2% (2T KISS)
  - 10,000 active wallets: up to 2% (2T KISS)
  - 100,000 active wallets: up to 2% (2T KISS)
  - 1,000,000 active wallets: up to 2% (2T KISS)
- **On-chain audit trail** — every release from the reserve is a visible transaction with a memo explaining the purpose
- **No silent releases** — the community is notified before any reserve tokens enter circulation

Why 100 Trillion Supply?

Bitcoin’s 21 million supply creates a denomination problem — buying a $4 coffee

costs 0.00004 BTC. Nobody thinks in five-decimal fractions. KISS takes the opposite approach: **everyday amounts should be large, satisfying, whole-ish numbers.**

Market Cap	Price per KISS	$4 Coffee	$50,000 Car
$1 billion	$0.00000001	400,000,000 KISS	5 trillion KISS
$100 billion	$0.000001	4,000,000 KISS	50 billion KISS
$1 trillion (BTC level)	$0.00001	400,000 KISS	5 billion KISS
$10 trillion (gold level)	$0.0001	40,000 KISS	500 million KISS

Even at gold-level market cap, a coffee is **40,000 KISS** — never tiny fractions. And sending it is always free.

Why 3 Decimals?

The sweet spot between simplicity and precision: - **3 decimals + 100 trillion supply fits in uint64** (184× headroom) — native CPU math, 8 bytes per balance - More precision than 2 decimals for sub-unit pricing flexibility - Still human-readable: `5000.500` not `0.000000000005` - No wei/gwei/ether confusion — what you see is what you send

---

6. Custom Token Standard — KISS-1

One token standard. One interface. No variants.

Create a Token

```
curl -X POST https://node.kisschain.org/v1/transaction \
  -d
'{"from":"kiss1abc...","to":"kiss1token_factory","amount":"100.000","memo":"C
```

The `memo` field handles everything. Token operations are just transactions with structured memos: - `CREATE:name:symbol:supply:decimals` - `SEND:symbol:amount` - `BURN:symbol:amount`

No bytecode deployment. No ABI encoding. No contract addresses. Tokens are a native feature, not a smart contract.

KISS Stablecoins — USDK, EURK, GBPK

The KISS-1 standard enables **fiat-pegged stablecoins** as first-class tokens on the KISS chain:

Token	Pegged To	Use Case
USDK	US Dollar	Global payments, trading pairs, DeFi base
EURK	Euro	European commerce, EUR-denominated payments
GBPK	British Pound	UK commerce
GOLDK	Gold (per oz)	Store of value, commodity peg

Creating a Stablecoin

```
          (#cb25-1)curl -X POST https://node.kisschain.org/v1/transaction \
          (#cb25-2)  -d
'{"from":"kiss1issuer...","to":"kiss1token_factory","amount":"1000.000","memo
```

One transaction. No smart contract audit. No Solidity. No deployment scripts.

Why This Matters

- **Stablecoins drive 70%+ of all crypto transaction volume** — without them, a chain is a hobbyist project
- **Zero-fee stablecoin transfers** — send $1 or $1M, no fee. KISS stablecoins become the lowest-cost stablecoin transfer layer in crypto
- **Simpler than ERC-20** — no approve/transferFrom patterns, no reentrancy bugs, no proxy upgrade complexity
- **Stablecoin volume strengthens the network** — more users, more stake, more relay activity; deflationary pressure comes from block-reward burn (10%), not transaction fees

Stablecoin Issuer Requirements

Stablecoin issuers on KISS must: 1. **Register as a verified issuer** (on-chain identity linked to legal entity) 2. **Provide proof of reserves** (attestation published monthly, verifiable on-chain) 3. **Maintain 1:1 backing** in fiat held at regulated custodians 4. **Support redemption** — holders can redeem tokens for fiat through the issuer

The KISS Foundation may issue the first USDK as a reference implementation, then open the standard to licensed third-party issuers.

KISS-1 AML Compliance — Built for Mainstream Adoption

KISS was designed to be the **“most mainstreamest”** blockchain — meaning Binance, Coinbase, and Kraken can list KISS-1 tokens without building custom compliance infrastructure. The `token_transfers` table satisfies **OFAC, FATF Travel Rule, and SAR requirements** out of the box.

Token Transfer Audit Trail

Every KISS-1 token SEND and BURN operation is logged to the `token_transfers` table:

```
Schema: (id, action, symbol, from_addr, to_addr, intl_id, amount, tx_hash, timestamp, block_height)
```

Field	Purpose
`action`	`SEND` or `BURN` — distinguishes transfers from destruction events
`from_addr` / `to_addr`	Full sender/recipient traceability (BURN has `to_addr=""`)
`intl_id`	Sender's INTL.ID compliance identity (if provided) — no join needed
`tx_hash`	Links to the originating KISS transaction for full context
`timestamp` / `block_height`	Immutable record of when the transfer occurred

Indexed on: `from_addr+symbol`, `to_addr+symbol`, `symbol`, `tx_hash` — O(1) lookup for compliance queries.

Exchange Compliance — Cross-Symbol Queries

Most blockchains require exchanges to track every token separately. KISS provides **cross-symbol AML queries** via a single API endpoint:

```
(#cb27-1)# Show me ALL token activity (any symbol) for this address
(#cb27-2)GET /v1/address/{address}/transfers?direction=received
(#cb27-3)
(#cb27-4)# Filter by specific token if needed
(#cb27-5)GET /v1/address/{address}/transfers?
symbol=USDK&direction=received
```

Compliance Requirements Satisfied

Requirement	How KISS-1 Satisfies It
OFAC Sanctions Screening	Indexed `to_addr` lookup — check if incoming transfer sender is on OFAC SDN list in O(1) time
FATF Travel Rule	`from_addr`, `to_addr`, and `intl_id` all in one record — full sender/recipient identity trail
SAR Filing	BURN events logged as `action=BURN` — detect suspicious destruction of tokens to cover tracks
Audit Trail	Immutable, indexed, paginated transfer history — regulator-ready
Exchange Deposit Detection	Query all incoming transfers across all tokens in one API call — no per-token infrastructure

Cursor-Based Pagination

The `token_transfers` API supports `before_id` cursor pagination:

```
(#cb28-1)# Get first 100 transfers
(#cb28-2)GET /v1/token/USDK/transfers/{address}?limit=100
(#cb28-3)
(#cb28-4)# Get next 100 using last ID as cursor
(#cb28-5)GET /v1/token/USDK/transfers/{address}?
limit=100&before_id=54321
```

Scales to millions of transfer records without degradation.

Why This Matters for Exchanges

Binance, Coinbase, and Kraken require: - Sender/recipient tracking for every token transfer - Sanctions screening (OFAC SDN list checking) - Audit trail for regulator inquiries - Ability to detect suspicious patterns (e.g., rapid burns after large transfers)

Most blockchains force exchanges to: - Run their own indexer nodes - Build

custom token-tracking infrastructure per standard (ERC-20, SPL, TRC-20) - Parse raw transaction logs to reconstruct transfer history - Maintain separate databases for compliance queries

KISS provides this natively: - One REST API call returns full transfer history - Works for any KISS-1 token without modification - No custom indexer required - No log parsing - No per-token infrastructure

Result: KISS-1 tokens are **cheaper to list** than ERC-20 tokens. Lower integration cost = more exchange listings = more liquidity = mainstream adoption.

---

7. How to Acquire KISS

All 100 trillion KISS exist at genesis — there is no ongoing minting. Here's how people get KISS at each stage:

At Launch

Method	How	Who
Buy with card (Stripe)	Buy KISS with credit/debit card on kisschain.org	Anyone worldwide
Public token sale	Buy KISS with USD, ETH, or BTC on kisschain.org	Anyone worldwide
Faucet	Free KISS for new wallets (see below)	New users
Ecosystem grants	Apply to build on KISS, receive grant in KISS	Developers, projects

After Launch (Ongoing)

Method	How	Difficulty
Buy with card	Visa/Mastercard via Stripe on kisschain.org or in-app	Easiest — like buying anything online
Exchanges	Buy on Coinbase, Binance, Kraken, etc.	Easy — same as buying any crypto
DEX swaps	Swap ETH/USDT/etc. for KISS on decentralized exchanges	Moderate
Relay earning	Run the phone app with relay mode on	Easy — toggle and forget
Staking rewards	Delegate KISS to a validator, earn ~5% annually	Easy — one tap in app
Accept as payment	Show QR code, receive KISS for goods/services	Easy
Peer-to-peer	Someone sends you KISS directly	Easiest

Buy KISS with a Card — Stripe Integration

Most crypto requires users to open an exchange account, complete KYC, deposit funds, and navigate a trading UI. KISS skips all of that:

```
1. Visit kisschain.org/buy (or tap "Buy KISS" in the app)
2. Enter amount in USD/EUR/GBP
3. Pay with Visa/Mastercard via Stripe
4. KISS delivered to your wallet in seconds
```

How It Works

```
User pays $10 via Stripe
  └── kisschain.org receives fiat
       └── KISS Foundation treasury releases equivalent KISS
            └── Sent directly to user's kiss1... address
                 └── Confirmed in ~10 seconds
```

Why This Is a Massive Advantage

	Most crypto projects	**KISS**
Fiat on-ramp	Months to get payment processor approval	Stripe already approved
User experience	Sign up for exchange → KYC → deposit → trade → withdraw	Enter amount → pay → done
Payment methods	Bank transfer (slow), crypto swap	Credit card, debit card, Apple Pay, Google Pay
Time to first KISS	30 minutes to days	**Under 60 seconds**
Available at launch	Usually no — requires exchange listing	Day 1

In-App Purchase

The KISS Wallet app includes a “Buy KISS” button powered by Stripe:

```
┌──────────────────────────────────────┐
│           Buy KISS                   │
│                                      │
│   Amount: [ $10.00 ] USD             │
│                                      │
│   You receive: ≈ 10,000,000 KISS     │
│   (rate updates live)                │
│                                      │
│   [ Pay with Card ]                  │
│   [ Apple Pay ]                      │
│   [ Google Pay ]                     │
│                                      │
│   Powered by Stripe                  │
└──────────────────────────────────────┘
```

No exchange account. No KYC beyond Stripe’s standard card verification. No seed phrase required before buying — the app creates a wallet automatically.

Pricing & Treasury

- **KISS Foundation** holds a treasury allocation (from genesis reserve)
- **Price** is set by market rate once exchanges list KISS; before that, set by the foundation
- **Stripe fees** (~2.9% + $0.30) absorbed by buyer or split — standard card processing
- **Anti-abuse:** Purchase limits per card (e.g., $10,000/day) to prevent wash trading

KISS Faucet — Free Onboarding

New users shouldn’t need to buy crypto before they can try it. The **KISS Faucet** gives every new wallet enough KISS to experiment:

```
1. Visit faucet.kisschain.org (or tap "Get Free KISS" in app)
2. Create or link an INTL.ID profile (free, one-time identity verification)
3. Create wallet (one click / one tap)
4. Receive 100,000 KISS for free
5. That's enough to meet the minimum send balance — start sending, staking,
and relaying immediately
```

Faucet Economics

	Value
Per user	100,000 KISS (meets minimum send balance)
Funded from	Ecosystem Grants pool (10% of supply = 10 trillion KISS)
Users onboardable	100,000,000 (100 million users)
Anti-abuse	One claim per verified INTL.ID profile (Sybil-resistant, zero cost to KISS Foundation)
Cost to KISS Foundation	$0 at launch (KISS has no market value yet)

The faucet is the **growth engine**. Every faucet claim creates a user who can immediately transact, relay, and stake — no exchange account, no credit card required. The INTL.ID gate prevents mass-claiming (one verified identity = one faucet claim) at zero per-user cost to the KISS Foundation, while simultaneously growing the INTL.ID user base.

---

8. Node Implementation

Target Specs

Property	Target
Language	Go (single binary, cross-platform)
Binary size	< 20 MB
Dependencies	Zero external (embedded DB, no Docker required)
Database	Embedded SQLite or BadgerDB (no Postgres/Redis to install)
Install	`curl -sSL install.kisschain.org | sh`
Run	`kiss node start`

Minimum Node Requirements

Resource	Full Node	Validator Node
CPU	1 core (any VPS)	2 cores
RAM	256 MB	512 MB
Disk	5 GB (year 1), ~10 GB/year growth	Same
Bandwidth	~10 GB/month	~25 GB/month
OS	Any Linux (Ubuntu 20.04+), macOS, Windows	Ubuntu 22.04+ recommended
Estimated cost	**$5/month VPS**	**$10/month VPS**

For comparison:

Chain	Validator Requirements	Monthly Cost
Bitcoin	4+ cores, 2 GB RAM, 700 GB SSD	~$40-80
Ethereum	4+ cores, 16 GB RAM, 2 TB NVMe	~$100-200
Solana	24+ cores, 128 GB RAM, 2 TB NVMe	~$500-1,000
KISS	1-2 cores, 256-512 MB RAM, 5 GB	**$5-10**

Why Embedded Databases Are Sufficient

- **SQLite** — Powers billions of devices (Firefox, Android, iOS). Handles millions of rows. WAL mode supports concurrent reads.
- **BadgerDB** — Go-native LSM-tree. Used by Dgraph in production. Fast key-value lookups.
- KISS state is just an **address → balance map** + transaction log. No contract state, no storage tries, no account nonces. A simple DB handles this trivially.

Integer Design — 64-bit Is Plenty

All balances and amounts use `uint64` (8 bytes) representing the smallest unit (0.001 KISS = 1 milliKISS):

	Value
Max uint64	18,446,744,073,709,551,615
Max KISS expressible	18,446,744,073,709,551.615 KISS (~18.4 quadrillion)
Total supply	100,000,000,000,000.000 KISS (100 trillion)
Headroom	184× total supply

Native 64-bit CPU arithmetic — no big-number libraries, no overflow risk, 4x less storage per balance vs 256-bit. 184× headroom means the supply could grow 184x before ever approaching uint64 limits — decades of staking inflation won’t come close.

Performance — Transaction Speed

	Bitcoin	Ethereum	Solana	KISS
Block time	~10 min	~12 sec	~400ms	**10 sec**
Finality	~60 min (6 conf)	~13 min (64 slots)	~13 sec (confirmed)	**10 sec (1 block, absolute)**
TPS (theoretical)	~7	~30	~65,000	**~1,000**
TPS (real-world)	~4	~15	~4,000	**~500-1,000**
“Safe to accept” time	30-60 min	5-15 min	5-15 sec	**10 sec**

Why ~1,000 TPS Is Enough

Without smart contracts, KISS doesn’t have: - DEX arbitrage bots flooding the mempool - NFT mint stampedes generating thousands of TPS - MEV searchers creating spam transactions - Contract-to-contract call chains multiplying execution

1,000 TPS = ~86 million transactions/day. Visa averages ~1,700 TPS globally. KISS covers real-world payment volume without Solana-level hardware requirements.

The Trade-off (Honest)

KISS trades peak throughput for **simplicity and accessibility**. You can’t run a Solana validator on a $5 VPS. You can run a KISS full node on a Raspberry Pi.

Why Thousands of Nodes Matter

The $5 node / Raspberry Pi target isn’t just a nice-to-have — it’s the **core security model**.

Censorship Resistance

Scenario	Solana (few hundred validators)	KISS (10,000 nodes in 80+ countries)
Government subpoena	Knock on ~20 data center doors → network at risk	10,000 operators across 80 jurisdictions → practically impossible
Regulatory shutdown	Concentrated in US/EU data centers → vulnerable	Apartments, garages, universities worldwide → unkillable
Coordinated DDoS	Small target set → feasible	Massive surface area → impractical

When only companies can afford validators, the network is a **permission slip away from shutdown**. When hobbyists run nodes on hardware they already own, the network becomes **sovereign**.

Trust Through Verification

Every node independently verifies every transaction. Users don’t have to trust a foundation, a company, or an exchange — they run a $50 Raspberry Pi and **verify the chain themselves**. This is the original promise of Bitcoin that most chains abandoned for performance.

Network Resilience

- **Solana:** 7+ network outages since launch. Small, hardware-intensive validator set = fragile.
- **Bitcoin:** Zero downtime since 2013. Tens of thousands of cheap nodes = indestructible.
- **KISS:** Targets the Bitcoin resilience model. So cheap to operate that the network can’t die.

Geographic Distribution

Cheap nodes = nodes everywhere. Not just AWS us-east-1 and eu-west-1, but: - An apartment in Nairobi - A garage in São Paulo - A university lab in Seoul - A Raspberry Pi in someone’s closet in Tokyo

More geography = lower latency globally + resilience against regional outages, natural disasters, and political events.

Community Ownership

When running a node costs $500/month, only businesses participate. When it costs **$0** on hardware you already own, **individuals** participate. This creates genuine grassroots ownership — people who run nodes care about the network personally, not as a line item on a corporate budget.

The “1,000 True Nodes” Thesis

You don’t need 10,000 to be resilient. **1,000 independently operated nodes in 50+ countries** makes a network practically indestructible: - No single legal jurisdiction covers them all - No DDoS can reach them all simultaneously - No hardware failure takes out more than a fraction - No corporate decision can pull the plug

The KISS design makes 1,000+ nodes not just possible, but **inevitable** — because the barrier to entry is a device most people already own.

AI-Deployable Node — Under 5 Minutes

KISS nodes are designed to be deployed by **humans, scripts, or AI agents** with zero configuration:

5 Ways to Run a Node — Pick Your Favorite

Every option gets you to a running node. Choose whichever feels natural:

Option 1: One-Line Install (recommended)

```
(#cb33-1)curl -sSL https://install.kisschain.org | sh
(#cb33-2)kiss node start
```

Downloads the Go binary, installs it, done. No dependencies. Works on any Linux, macOS, or WSL.

Option 2: Docker (one command)

```
(#cb34-1)docker run -d --name kiss-node -p 7700:7700 -v kiss-data:/data ghcr.io/kisschain/kiss-node:latest
```

That’s it. Container pulls, starts, syncs. Data persists in a Docker volume.

```
(#cb35-1)# Check status
(#cb35-2)docker logs kiss-node
(#cb35-3)
(#cb35-4)# Stop
(#cb35-5)docker stop kiss-node
(#cb35-6)
(#cb35-7)# Upgrade
(#cb35-8)docker pull ghcr.io/kisschain/kiss-node:latest && docker restart kiss-node
```

Option 3: Docker Compose (for the tidy)

```
(#cb36-1)# docker-compose.yml
(#cb36-2)version: "3"
(#cb36-3)services:
(#cb36-4)  kiss-node:
(#cb36-5)    image: ghcr.io/kisschain/kiss-node:latest
(#cb36-6)    container_name: kiss-node
(#cb36-7)    ports:
(#cb36-8)      - "7700:7700"
(#cb36-9)    volumes:
(#cb36-10)      - kiss-data:/data
(#cb36-11)    restart: unless-stopped
(#cb36-12)volumes:
(#cb36-13)  kiss-data:
```

```
(#cb37-1)docker compose up -d
```

Option 4: Download Binary from GitHub

```
(#cb38-1)# Download latest release
(#cb38-2)wget
https://github.com/kisschain/kiss/releases/latest/download/kiss-linux-amd64
(#cb38-3)chmod +x kiss-linux-amd64
(#cb38-4)mv kiss-linux-amd64 /usr/local/bin/kiss
(#cb38-5)kiss node start
```

Releases published for every platform:

Binary	Platform
`kiss-linux-amd64`	Ubuntu, Debian, most VPS
`kiss-linux-arm64`	Raspberry Pi, ARM servers
`kiss-darwin-amd64`	macOS (Intel)
`kiss-darwin-arm64`	macOS (Apple Silicon)
`kiss-windows-amd64.exe`	Windows

Option 5: AI Agent (fully automated)

An AI agent (ChatGPT, Claude, Cascade, etc.) can deploy a KISS node end-to-end:

```
Step 1: Provision VPS via API                          → 2 minutes
        (DigitalOcean, Hetzner, Vultr — any $5 plan)
Step 2: SSH in, run install script                     → 1 minute
Step 3: kiss node start                                → 10 seconds
Step 4: Verify: kiss node status                       → Done
```

Total: under 5 minutes, zero human intervention.

One-Click Deploy (Web UI)

For non-technical users, kisschain.org offers a deploy button:

```
1. Click "Launch a Node"
2. Connect your cloud account (DigitalOcean / Hetzner)
3. Choose region
4. Click "Deploy"
→ Node running in under 5 minutes, $5/month
```

Why This Matters

Every other blockchain requires a tutorial, a Discord channel, and hours of troubleshooting to run a node. KISS nodes are **boring to set up** — and that’s the point. The fewer barriers, the more nodes. The more nodes, the stronger the network.

Zero-Configuration Design

Setting	Other chains	KISS
Config file	50+ settings, YAML/TOML	None — sane defaults, override with flags if needed
Genesis file	Manual download, checksum verify	Embedded in binary
Database	Install Postgres/Redis separately	Embedded, auto-created
Ports	Manual firewall config	Auto-discovers available port, UPnP
Peer discovery	Bootstrap nodes list, manual DNS	Hardcoded seed nodes + automatic peer exchange
Upgrades	Manual download, restart, pray	`kiss node upgrade` or `docker pull`

GitHub Repository — github.com/kisschain/kiss

Open source from day one. The entire project lives in a single repository:

```
kisschain/kiss/
├── README.md                  Project overview + quickstart
├── WHITEPAPER.md              Full technical specification
├── LICENSE                    MIT License
├── Dockerfile                 Single-stage Docker build
├── docker-compose.yml         One-command Docker setup
├── Makefile                   build, test, install targets
├── go.mod                     Go module (zero external deps where possible)
│
├── cmd/
│   └── kiss/                  CLI entry point
│       └── main.go            14 commands, single binary
│
├── internal/
│   ├── chain/                 Block + transaction processing
│   ├── consensus/             DPoS round-robin validator logic
│   ├── crypto/                BLAKE3 hashing, Ed25519 signing
│   ├── db/                    Embedded SQLite/BadgerDB storage
│   ├── api/                   REST API (8 endpoints + WebSocket)
│   ├── p2p/                   Peer-to-peer networking (libp2p)
│   ├── relay/                 Light relay node protocol
│   ├── wallet/                Key generation, HD derivation
│   └── token/                 KISS-1 token standard
│
├── scripts/
│   └── install.sh             The one-line installer
│
├── explorer/                  Block explorer (single-page web app)
│
└── tests/
    ├── chain_test.go
    ├── api_test.go
    └── ...
```

Key Principles

- **One repo** — not 15 repos across 3 GitHub orgs. Everything in one place.
- **One binary** — `go build ./cmd/kiss` produces a single executable. That's the node, the CLI, and the wallet.
- **One Dockerfile** — multi-stage build, final image < 30 MB.
- **MIT License** — fully open source. Fork it, modify it, run it. No restrictions.
- **No monorepo tooling** — no Bazel, no Nx, no Turborepo. Just `go build` and `go test`.

CLI — Entire Command Set

```
kiss wallet create                Create a new wallet (default account)
kiss wallet create --account N    Derive HD address for account N
kiss wallet list                  List all derived addresses
kiss wallet balance               Check balance
kiss send <to> <amount>           Send KISS
kiss transaction <hash>           Look up transaction
kiss node start                   Start a node
kiss node status                  Node status
kiss node upgrade                 Upgrade to latest version
kiss stake <amount>               Stake KISS
kiss delegate <val> <amt>         Delegate to validator
kiss relay start                  Start phone/light relay mode
kiss relay status                 Relay stats and earnings
kiss version                      Show version
```

14 commands. That's the entire CLI.

---

9. Participation Model — Three Tiers

Not everyone can run a server. KISS offers three ways to participate and earn rewards, from full validator to phone in your pocket:

Tier 1: Full Validator

Device	VPS ($5-10/month), Raspberry Pi, any server
What it does	Produces blocks, validates all transactions, stores full chain
Earns	Full staking rewards (5% annual on staked amount)
Minimum stake	1,000,000,000 KISS (1 billion)
Setup	`curl -sSL install.kisschain.org | sh && kiss node start`

Tier 2: Light Staker (Delegation)

Device	Phone, laptop, tablet — anything
What it does	Delegates KISS to a validator, monitors chain with light client
Earns	Delegation rewards (validator's 5% minus their commission, typically 5-10%)
Minimum stake	1 KISS (no minimum — delegate any amount)
Setup	Open app → tap "Stake" → choose validator → enter amount → done

Tier 3: Relay Node (Phone "Mining")

This is the **phone-friendly participation layer** — the closest thing to mobile mining, but useful and battery-friendly.

Device	Android phone, iPhone, any device with internet
What it does	Receives and forwards transactions across the network (gossip relay)
Earns	Small relay reward per forwarded transaction
Minimum stake	None — relay is free to run
Battery impact	Minimal — network traffic only, no CPU-intensive computation
Setup	Open app → toggle “Relay Mode” → earning

How Relay Rewards Work

```
1. Your phone receives a new transaction from a nearby node
2. Your phone forwards it to other connected nodes
3. When that transaction is confirmed in a block, you earn a small relay fee
4. Reward: 1 KISS per relayed transaction (funded from block rewards)
5. Phone relays ~100-500 tx/hour passively = 100-500 KISS/hour
```

Why Relay Nodes Are Valuable

- **More relays = faster transaction propagation** — transactions reach validators quicker
- **More relays = more geographic coverage** — mobile devices move, creating mesh coverage
- **More relays = harder to partition the network** — phones are everywhere
- **Zero cost to the relay operator** — just background network traffic

Comparison: Phone “Mining” Across Chains

	Bitcoin Mining	Ethereum Staking	Pi Network	KISS Relay
Phone viable?	Impossible	Need 32 ETH + server	⚠ Questionable value	Genuinely useful
Battery drain	N/A	N/A	Moderate	Minimal
Actually helps network?	N/A	N/A	Debatable	Faster tx propagation
Earnings	$0	N/A	Unclear	Small but real
Complexity	N/A	N/A	App + KYC + referrals	Toggle on/off

Reward Distribution Per Block

Source	% of Block Reward	Goes to
Validators	70%	Block producer + delegators (proportional to stake)
Relay nodes	20%	Distributed to relays that propagated the block’s transactions
Burn	10%	Destroyed (fixed deflationary pressure from block rewards — independent of transaction fees)

10. Mobile App — KISS Wallet

One app. Wallet + relay + staking. Available on Android and iOS.

Full Self-Custody — You Own Your KISS

The KISS Wallet is a **full self-custody wallet.** The private key is generated on your phone, stored in the hardware secure enclave (Android Keystore / iOS Keychain), and **never transmitted anywhere.** Not to KISS servers, not to anyone. The phone signs transactions locally and broadcasts them.

You own your KISS. Period. No exchange, no foundation, no company in the middle.

	Self-Custody (KISS Wallet)	**Custodial (e.g., Coinbase app)**
Who holds private keys?	You — on your phone's secure hardware	The company
Can they freeze your funds?	No — impossible	Yes
Need permission to send?	No	Technically yes
If company shuts down?	Your KISS is fine	You might lose access
Backup responsibility	You save your seed phrase	They handle it

Wallet Lifecycle

```
1. CREATE   → Key generated on device, stored in secure enclave
2. BACKUP   → Shown a 12-word seed phrase (Ed25519-derived)
              Write it down, store it safely
3. SEND     → Phone signs the transaction locally
              Broadcasts to nearest relay/node
4. RECEIVE  → Share your kiss1... address or QR code
5. RECOVER  → Install app on new phone
              Enter seed phrase → wallet fully restored
```

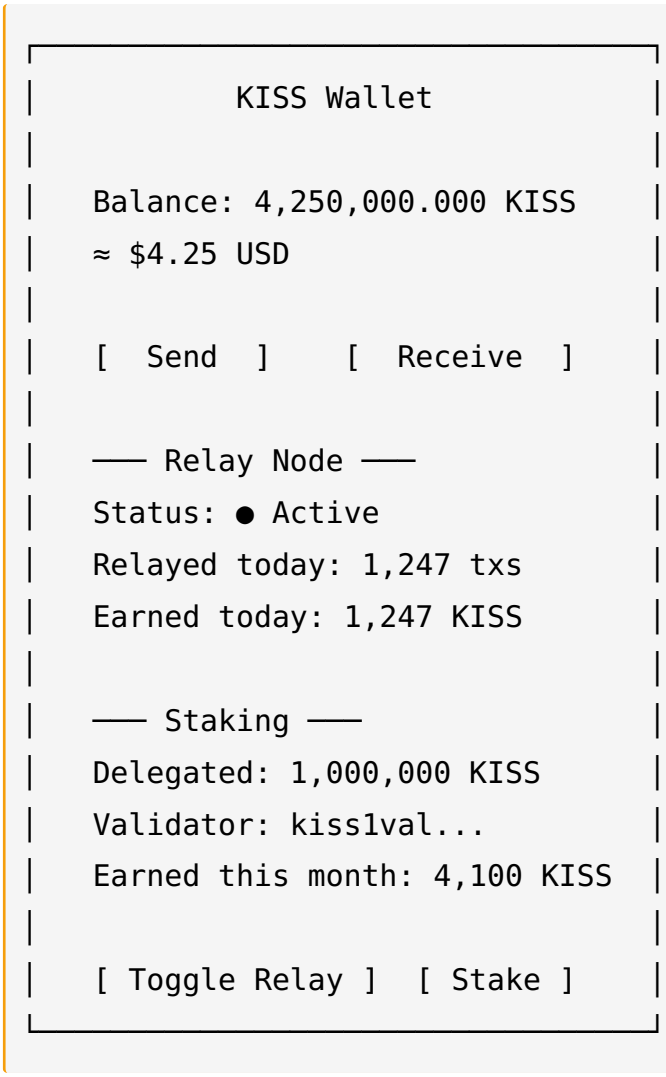
No accounts. No passwords. No email. No KYC. Just a key on your phone and a seed phrase backup. **That’s KISS.**

How Signing Works (Under the Hood)

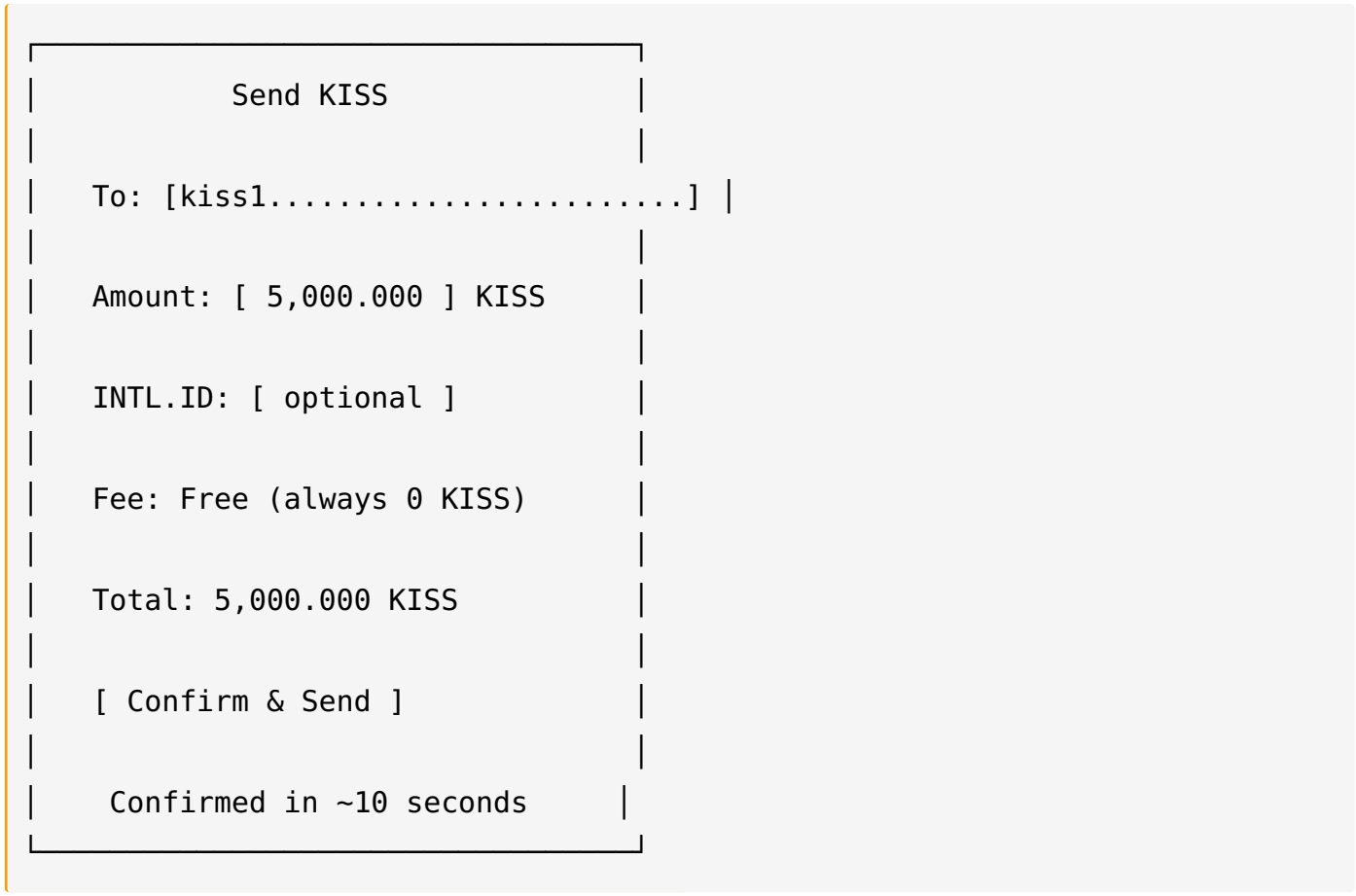
```
You tap "Send 5,000 KISS"
  └── App builds transaction JSON
       └── Secure enclave signs it with your Ed25519 private key
            └── Private key NEVER leaves the enclave
                 └── Signed transaction broadcast to network
                      └── Confirmed in ~10 seconds
```

The private key is stored in dedicated security hardware on your phone — the same chip that protects Face ID and fingerprint data. Even if the phone is compromised, the key cannot be extracted.

Screen: Home


KISS Wallet
Balance: 4,250,000.000 KISS
≈ $4.25 USD
[Send]
[Receive]
Relay Node
Status: ● Active
Relayed today: 1,247 txs
Earned today: 1,247 KISS
Staking
Delegated: 1,000,000 KISS
Validator: kiss1val...
Earned this month: 4,100 KISS
[Toggle Relay]
[Stake]

Screen: Send



Design Principles

- **Two main actions:** Send and Receive. Everything else is secondary.
- **No settings page.** Nothing to configure.
- **No network selector.** There’s one KISS network. Period.
- **No gas estimation.** No fee. Amount sent = amount received.
- **No seed phrase on first launch.** Auto-generated, backed up when user is ready.
- **Relay toggle:** One tap to start earning. One tap to stop.

Tech Stack

Component	Technology
Framework	React Native (Android + iOS from one codebase)
Key storage	Device secure enclave (Keystore on Android, Keychain on iOS)
Light client	Embedded — verifies block headers, not full blocks
Relay protocol	libp2p (same as full nodes, lightweight mode)
Size	Target < 20 MB install

---

11. Security Model

Threat Matrix — Honest Assessment

Threat	Attack Method	Defense	Effectiveness
Script kiddie spam	Flood transactions from one wallet	Balance-based send rate limits + per-IP rate limiting	✅ Fully mitigated — must buy and hold KISS (capital lockup), can't easily resell at low market cap
Botnet spam	Distributed flood from thousands of IPs	Balance-based rate limits + mempool cap	✅ Each wallet independently rate-limited regardless of IP
Competitor attack	Sustained spam to degrade network	Rate limits + block size cap + congestion-mode ordering	✅ Rate-capped, congestion pushes spam to back of queue, legitimate commerce flows
		Rate limits —	⚠ Partially mitigated

State-level spam	Government-funded transaction flood	caps throughput regardless of attacker budget	(rate limits cap throughput; congestion-mode prioritizes real commerce)
Regulatory ban	Outlaw KISS in a jurisdiction	10,000+ nodes in 80+ countries	Network survives
ISP blocking	Block KISS traffic at network level	Tor/VPN support for node peering	Circumventable
Operator arrest	Target known node operators	Anonymity + sheer volume of operators	Too many to target
51% stake attack	Acquire majority of staked KISS	Max 100 validators + delegation spread	⚠ Depends on stake distribution

Four Layers of Defense

```
Layer 1: CAPITAL LOCKUP (economic)
  → Zero transaction fee — but minimum 100,000 KISS balance required to send
  → Spam requires buying and holding KISS. The capital is locked during the attack.
  → At low market caps, thin liquidity makes it hard to exit (sell) after a spam campaign
  → Fee framework retained: governance can activate fees via software upgrade if ever needed
  → Effective against: bots, competitors, any attacker who must acquire capital to spam
```

```
Layer 2: BALANCE-BASED SEND RATE LIMITS (protocol)
  → MINIMUM BALANCE TO SEND: 100,000 KISS — below this, wallet can receive
but not send
  → 100K KISS balance: 1 tx / 5 min (default tier)
  → 10M KISS balance: 1 tx / min
  → 100M KISS balance: 1 tx / 10 sec
  → 1B KISS balance: 1 tx / sec
  → Receiving is UNLIMITED — only sending is rate-limited
  → Minimum balance caps wallet splitting: 1B KISS = max 10,000 wallets
(not 1M dust wallets)
  → Forces spammers to buy and hold KISS — creates economic friction +
capital lockup
  → At low market cap, thin liquidity means spammers can't easily exit
their position
  → Effective against: script kiddies, bots, any attacker at any market cap

Layer 3: NETWORK RATE LIMITS (mempool)
  → Per-IP: max 10 transactions/second
  → Mempool cap: reject transactions when queue is full
  → CONGESTION MODE: when mempool ≥ 80% full, block ordering switches from
FIFO to
    amount-descending — larger transactions processed first, dust spam
pushed to back
  → Reverts to FIFO when mempool drops below 60%
  → Zero governance parameters — purely relative ordering, self-adjusting
at any price
  → Effective against: raw network floods, single-source attacks, dust spam
during congestion

Layer 3.5: IP REPUTATION SCORING (node-level)
  → Each node tracks unique sender addresses per IP over a rolling 10-
minute window
  → If one IP submits transactions for 100+ unique addresses → deprioritize
(not reject)
```

```
  → Hardcoded threshold — not configurable, not a governance parameter
  → In-memory only, ephemeral, resets on node restart. No exemptions list.
  → Not a consensus rule — each node applies locally to its own mempool
  → Effective against: single-VPS spammers controlling thousands of wallets

Layer 4: DECENTRALIZATION (political)
  → 10,000+ nodes across 80+ countries
  → $5/month to run = massive operator base
  → No single jurisdiction can shut down the network
  → Effective against: governments, regulators, censorship
```

Why Zero Fees + Rate Limits Work

The defense is **capital lockup + throughput capping**, not fee depletion:

Market Cap Stage	Fee Defense	Capital Lockup + Rate Limits	Combined Effect
Early (low value)	None — fees meaningless ($0.00001 per tx)	Active — min balance caps wallets; rate limits cap throughput	Rate limits + capital lockup carry the defense
Growth	None — fees still zero	Active — capital cost rises as KISS appreciates	Capital lockup cost rises naturally with price
Mature (high value)	None — fees still zero	Active — 100K KISS minimum becomes substantial	Minimum balance becomes a meaningful capital commitment

Key insight: **the defense improves automatically as KISS appreciates.** A 100K KISS minimum costs $0.001 at $1M market cap and $100 at $1T market cap —

without any governance action. The market adjusts spam economics on its own.

Spam Economics

Single-wallet attack (attacker uses one wallet at the highest tier their balance allows):

Attacker Budget	KISS Acquired (at $1T mcap)	Max Spam Rate	Capital Locked	Attack Duration
$1	100K KISS	1 tx/5min	$1	Indefinite but bounded
$100	10M KISS	1 tx/min	$100	Indefinite but bounded
$1,000	100M KISS	1 tx/10sec	$1,000	Indefinite but bounded
$10,000	1B KISS	1 tx/sec	$10,000	Indefinite but bounded

With zero fees, spam attacks are indefinite — but they are bounded by rate limits and capital lockup. The attacker’s KISS cannot be used for anything else while attacking. At low market caps, thin liquidity makes it hard to exit the position after the attack.

Wallet-splitting attack (attacker splits into maximum 100K-minimum wallets):

Attacker Budget	KISS Acquired	Max Wallets (÷100K)	Max Spam Rate	vs. Network Capacity (1,000 TPS)
$1	100K KISS	1 wallet	12/hr	< 0.001%
$100	10M KISS	100 wallets	1,200/hr	< 0.1%
$1,000	100M KISS	1,000 wallets	12,000/hr	< 0.4%
$10,000	1B KISS	10,000 wallets	120,000/hr	~3.3%

The minimum balance is what caps the splitting attack. Without it, the $10,000 attacker could create 1M dust wallets for 12M tx/hr — **100× worse and enough to fill the network.** With the 100K minimum, the ceiling is 120K tx/hr (~3.3% of capacity), and congestion-mode ordering pushes all the dust to the back of the queue regardless.

The practical worst case: a well-funded spam campaign creates congestion but does not halt legitimate commerce — real transactions (with real amounts) get processed first by congestion-mode ordering.

Congestion-Mode: Amount-Based Transaction Priority

Under normal conditions, block ordering is **FIFO** — first-in, first-out. No priority, no fee auctions, no MEV.

During congestion (mempool ≥ 80% capacity), ordering switches to **amount-descending** — larger transactions are processed first:

```
Normal mode (mempool < 80% capacity):
  → FIFO — transactions processed in submission order

Congestion mode (mempool ≥ 80% capacity):
  → Amount-descending — larger transactions first
  → Ties broken by submission time (FIFO within same amount)
  → Reverts to FIFO when mempool drops below 60% capacity
```

This pushes dust spam to the back of the queue while legitimate commerce (which moves real value) flows through. An attacker can attempt to game this by self-transferring large amounts, but doing so requires funding each wallet with more capital — directly reducing the number of wallets they can create. The attacker must choose between **volume** (many dust transactions, deprioritized) and **priority** (few large transactions, but far fewer of them). The minimum balance requirement further constrains self-transfer loops: a wallet with exactly 100K KISS can send all 100K, but the recipient starts with exactly 100K — any further sends would drop them below the minimum. The loop is effectively limited to one hop of capital movement.

Properties: - **Zero governance parameters** — purely relative ordering, self-adjusting at any KISS price - **Only active during congestion** — FIFO is the default; users never notice unless the mempool is under pressure - **No MEV risk** — ordering is deterministic (amount descending, then timestamp); validators cannot reorder for profit - **Backwards-compatible** — no new transaction fields required

Why Zero Fees Don’t Weaken Security

At early market caps, KISS has near-zero monetary value — **no fee amount prevents spam on a worthless chain.** This is true for every cryptocurrency at launch. Bitcoin’s early blocks could have been spammed for free too.

Early-stage defense relies on **Layer 2 (balance-based send rate limits)**. Spammers must buy and hold KISS to send at any meaningful rate, and at low market caps, thin exchange liquidity makes it difficult to exit after a spam attack. As KISS gains value, the minimum balance requirement becomes a meaningful capital commitment on its own — without any fee at all.

The honest tradeoff: With zero fees, spam attacks are indefinite rather than self-exhausting. An attacker’s capital is locked but not consumed. The defense accepts this tradeoff because: (1) rate limits already cap the damage to a bounded, manageable level; (2) congestion-mode ordering ensures real commerce flows even during spam campaigns; and (3) the zero-fee property is a powerful adoption driver that outweighs the incremental security cost.

State-Level Adversaries

Governments don’t spam blockchains — it’s the most expensive and least effective attack vector. They use cheaper tools: - **Regulation:** Ban exchanges from listing KISS ($0 cost, high impact in one country) - **ISP blocking:** Order ISPs to block KISS traffic ($0 cost, circumventable with Tor) - **Arrest:** Target known operators ($0 cost, ineffective against 10,000 anonymous nodes)

The defense against state-level adversaries is **not the fee — it’s the node network.** A $5 Raspberry Pi in 80 countries is harder to kill than a $500 server in 5 data centers. This is why the lightweight node design is a security feature, not just a convenience.

---

12. Integration Guide — Exchanges, Wallets & Apps

KISS is designed to be the **easiest L1 blockchain to integrate.** No SDKs required — just REST + JSON.

Exchange Integration (Coinbase, Kraken, Binance)

What an exchange engineer needs to do:

Task	Ethereum	KISS
Run a node	4+ cores, 16 GB RAM, 2 TB NVMe, complex config	`kiss node start` — done
Detect deposits	Scan logs, handle reorgs, track confirmations	Scan blocks for `to` field. 1 block = final. No reorgs.
Send withdrawals	Build tx, estimate gas, manage nonce, handle EIP-1559	POST one JSON object. No fee.
Check balances	`eth_getBalance` + token contract calls	`GET /v1/balance/{address}`
Confirm transactions	Wait 12-64 blocks, handle uncle blocks	1 block = confirmed. Period.
KYC/AML compliance	Third-party chain analytics (Chainalysis, Elliptic)	Read `intl_id` field → verify via INTL.ID API
Support tokens	Implement ERC-20 ABI, each token different	Same `/v1/transaction` endpoint, read `memo` field

Estimated Integration Time

Chain	Typical Engineering Effort
Ethereum	2-4 months
Solana	3-6 months
KISS	**1-2 weeks**

Minimal Exchange Integration (3 endpoints)

```
# 1. Check customer balance
GET /v1/balance/{address}

# 2. Process withdrawal
POST /v1/transaction

{"from":"kiss1exchange...","to":"kiss1customer...","amount":"50000.000","memo


# 3. Confirm deposit
GET /v1/transaction/{hash}
→ {"ok": true, "data": {"confirmed": true, "block":
12345}}
```

That’s a complete exchange integration. Three endpoints. No SDK. No special libraries.

Wallet Integration (Third-Party Wallets)

Wallet Type	How to Support KISS	Effort
MetaMask	MetaMask Snap plugin (KISS is not EVM)	Moderate — different key curve (Ed25519 vs secp256k1)
Trust Wallet	Native integration via WalletConnect	Low — standard WalletConnect protocol
Ledger / Trezor	Ed25519 app for hardware signing	Moderate — Ed25519 already supported on both devices
Custom wallet	Call 3 REST endpoints	Trivial — any language, any platform

WalletConnect Support

KISS Wallet supports the **WalletConnect v2** protocol, enabling any web app to connect:

```
1. Web app shows QR code or deep link
2. KISS Wallet scans / opens link
3. User approves connection
4. Web app can request transactions (user confirms each one in app)
```

No browser extension required. No MetaMask dependency. Any website can accept KISS payments by integrating WalletConnect — a well-documented, widely-supported open standard.

Why KISS Is the Easiest Chain to Integrate

```
Ethereum:   eth_sendRawTransaction, eth_estimateGas,
eth_getTransactionReceipt,
            eth_getBlockByNumber, eth_getLogs, eth_call, eth_getCode,
            handle reorgs, handle uncle blocks, gas price oracle,
            nonce management, EIP-1559 fee logic, ABI encoding...

KISS:       POST /v1/transaction   → send
            GET  /v1/transaction   → confirm
            GET  /v1/balance       → check balance
            Fee: Zero. Always.
            Finality: 1 block. Always.
            Done.
```

---

13. Development Roadmap

Phase 1: Foundation (Q3 2026)

- ☐ Core data structures (blocks, transactions, addresses)
- ☐ BLAKE3 hashing + Ed25519 signing
- ☐ In-memory blockchain
- ☐ REST API (8 endpoints)
- ☐ CLI wallet
- ☐ Single-node testnet

Phase 2: Consensus (Q4 2026)

- ☐ P2P networking (libp2p, kept simple)
- ☐ Round-robin block production
- ☐ Staking / delegation
- ☐ Multi-node testnet
- ☐ Block explorer (single-page, minimal)

Phase 3: Tokens & Launch (Q1 2027)

- ☐ KISS-1 token standard
- ☐ Genesis distribution
- ☐ Public testnet

- ☐ Security audit
- ☐ Mainnet launch

Phase 4: Mobile & Relay (Q2 2027)

- ☐ KISS Wallet app (Android + iOS)
- ☐ Light client (verify headers, not full blocks)
- ☐ Relay node protocol
- ☐ Relay reward distribution
- ☐ One-click node deploy on kisschain.org

Phase 5: Ecosystem (Q3 2027+)

- ☐ Payment gateway integration (TheCashier.com)
- ☐ WalletIDs.net native KISS support
- ☐ Exchange listings
- ☐ AI-deployable node toolkit (API for automated provisioning)
- ☐ Block explorer (single-page, minimal)

---

14. Competitive Positioning

Chain	Philosophy	KISS Advantage
Bitcoin	Store of value, script-based	KISS: simpler tx model, instant finality, zero fees
Ethereum	World computer, unlimited flexibility	KISS: no VM complexity, predictable behavior
Solana	Maximum performance	KISS: no PoH complexity, simpler to run a node
Nano	Feeless, instant	KISS: also feeless + tokens + staking rewards + `intl_id` compliance + simpler API
Stellar	Simple payments	KISS: even simpler — no anchors, no path payments

KISS vs Nano — The Closest Competitor

Nano is the only other proven feeless L1, making it KISS’s most direct competitor. Both chains share the same founding principle — free transactions, instant finality — but diverge sharply on sustainability, features, and developer experience.

What They Share

Property	KISS	Nano
Transaction fee	Zero	Zero
Confirmation speed	~10 seconds (1 block)	~0.2–2 seconds (async DAG)
Smart contracts	No (by design)	No
Signatures	Ed25519	Ed25519
Node cost	~$5/month	~$10/month
Privacy	Pseudonymous	Pseudonymous

Where KISS Goes Further

Feature	KISS	Nano
Token standard	KISS-1 (memo-based)	None
Staking rewards	5% annual inflation	None
Validator incentives	Block rewards	Volunteer nodes only
Compliance layer	`intl_id` field	None
API simplicity	8 REST endpoints (`curl`)	JSON-RPC (complex)
Relay incentives	Gossip relay rewards	None
Stablecoins	KISS-1 token standard	Not possible
Battle-tested	Launching 2026	10+ years live

The Sustainability Problem

Nano's most significant structural weakness: **node operators earn nothing.** Representative nodes process transactions as a public service — zero income, real costs. This was a deliberate design choice, but creates compounding problems:

- **Node count is declining** — no economic incentive to run infrastructure
- **No attack-cost asymmetry** — spammers don't lose anything; neither do defenders
- **Foundation dependency** — Nano Foundation funds development from reserves; when reserves run out, development stops
- **No token ecosystem** — without a token standard, Nano cannot participate in the 70%+ of crypto volume driven by stablecoins and assets

KISS solves this directly: validators earn from a reserved 25% staking pool (~25 trillion KISS over 10 years). Nodes have a real economic reason to run reliably and upgrade promptly.

The Developer Experience Problem

Nano uses a JSON-RPC interface modeled on Bitcoin Core. To integrate, a developer must understand: - The **block-lattice model** — every transfer is two separate blocks (send + receive) - **Local PoW generation** — even feeless, each block requires proof-of-work for spam prevention - **Representative voting** — understanding voting weight, frontier blocks, epoch blocks

KISS reduces integration to: `POST /v1/transaction`, `GET /v1/transaction/{hash}`. No special model to learn. No PoW. No dual-block transfers. `curl` is the SDK.

Verdict

Nano proved the feeless model works and is viable at scale. KISS takes that proof and adds the missing pieces: validator incentives for long-term sustainability, a token standard for real-world volume, a compliance layer for institutional adoption, and an API simple enough to integrate in an afternoon.

Target Audience

1. **Developers** tired of toolchain complexity — build in an afternoon
2. **Businesses** that want crypto payments without a blockchain team
3. **Users** who want to send money, not manage gas settings
4. **Educators** teaching blockchain — KISS is the textbook example

Adoption Model: Consumer-Pull

Most L1 blockchains pursue top-down adoption — enterprise sales, hackathon sponsorships, paid integrations. This is slow, expensive, and produces fragile adoption that disappears when incentives end.

KISS is designed for **bottom-up, consumer-pull adoption:**

```
1. Consumer discovers KISS (faucet, app store, word of mouth)
2. Onboarding is instant — tap "Get Free KISS," receive 100K KISS, done
3. Sending is obvious — enter amount, tap Send, confirmed in 10 seconds
4. No gas, no hex, no seed phrase anxiety — it just works
5. Consumer pays friends in KISS → friends onboard → network grows
6. Consumer asks a business: "Do you accept KISS?"
7. Business checks integration docs → 3 API calls → ships that afternoon
8. Business now receives and sends KISS — adopted without a sales pitch
```

The strategic implication: **KISS doesn't need to convince businesses. It needs to be so simple that consumers adopt it organically, and businesses follow because customers demand it.**

This inverts the typical adoption priority:

Priority	Business-Push Model (Most L1s)	Consumer-Pull Model (KISS)
#1	Enterprise partnerships	Faucet UX, mobile app simplicity
#2	Developer grants, hackathons	Onboarding speed (zero to sending in 30 seconds)
#3	Exchange listings	Social sharing ("pay me in KISS")
#4	End-user marketing	Developer docs, 3-call integration guide

The faucet is the sales team. The mobile app is the pitch deck. The 10-second finality is the demo. The `intl_id` is the compliance answer that lets businesses say yes when customers ask.

When a customer says "pay me in KISS," the business doesn't need convincing — they need a quick integration guide and an afternoon.

---

15. Open Questions

- ☐ Should memo-based token creation be the permanent model or a v1 simplification?
- ☐ Privacy features — add optional privacy or stay fully transparent?
- ☐ Bridge to other chains — how to keep it simple while enabling interop?
- ☐ Governance mechanism — software upgrade (Bitcoin model) vs. validator voting (Cosmos model)? See Section 3.7. Conservative default: software upgrade.
- ☐ What's the minimum viable validator count for launch? (21? 50? 100?)
- ☒ INTL.ID on-chain validation? **Decision: No.** Chain stores the field, does not validate. Recipient verifies off-chain.
- ☒ INTL.ID verification levels (L1/L2) on-chain? **Decision: No.** Presence/absence of `intl_id` is the only signal the protocol needs. Verification depth is an INTL.ID platform concern, returned by their API.
- ☐ INTL.ID API specification — define the verification endpoint contract for recipients
- ☐ Should wallets auto-populate `intl_id` if the user has linked an INTL.ID profile?

---

16. Prior Art & Research

Project	What to learn from it
Bitcoin	UTXO simplicity, conservative design
Nano	Feeless UX, instant transactions
Stellar	Payment-focused chain, simple operations
MimbleWimble / Grin	Minimalist protocol design
Cosmos SDK	Modular chain building (but simplify further)

Document Version: 0.2 (Draft) Created: 2026-02-15 Updated: 2026-02-16 — Added `intl_id` protocol-native compliance identity, balance-based send rate limits, governance-adjustable fees (Section 3.7) Updated: 2026-02-19 — Transaction fee set to zero (free KISS). Fee framework retained, governance-adjustable. Spam defense via rate limits + minimum balance + congestion-mode ordering. Sections 3.3, 3.7, 5, 6, 9, 10, 11, 12, 14 updated. Author: NSDB / WalletIDs.net Team Next: Technical prototype — single-node chain with CLI + API