Contents

NexaFlow: A Confidential Payment Network with Trust-Based Credit, Deflationary Tokenomics, and Byzantine-Fault-Tolerant Consensus

NexaFlow Contributors

February 2026

Abstract

We present NexaFlow, a cryptocurrency protocol that combines an account-based ledger model with a UTXO-style confidential output layer to support both transparent and fully private payments on a single network. Transaction amounts, sender identity, and recipient identity can be hidden through Pedersen commitments, linkable ring signatures, and stealth addresses, while remaining cryptographically verifiable by all participants. A native deflationary token (NXF) employs a fee-burning mechanism as the sole source of supply reduction and a tiered staking system with dynamic interest as the sole source of new issuance—creating a self-balancing economic feedback loop. The network reaches agreement through a Byzantine-fault-tolerant variant of the Ripple Protocol Consensus Algorithm (BFT-RPCA), which delivers deterministic finality without proof-of-work mining. An embedded trust-line graph enables cross-currency credit relationships and multi-hop payment path finding, while an integrated limit-order matching engine provides decentralised exchange capabilities. All performance-critical cryptographic primitives are compiled from Cython to native C for throughput suitable for production workloads.

---

1. Introduction

The original Bitcoin whitepaper [1] demonstrated that a purely peer-to-peer electronic cash system could operate without a trusted third party by using proof-of-work to order transactions in an append-only ledger. Subsequent systems have explored alternative consensus mechanisms [2], privacy techniques [3][4], and credit-based payment networks [5] as complementary or competing design axes.

However, each axis has typically been pursued in isolation. Privacy-focused cryptocurrencies such as Monero [3] provide strong sender and amount confidentiality but lack native support for multi-currency credit. Consensus-oriented protocols like the XRP Ledger [2] offer fast finality and trust-line infrastructure but expose all transaction details on-chain. Staking-based systems provide economic incentives for participation but rarely integrate privacy or credit natively.

NexaFlow unifies these capabilities into a single coherent protocol:

1. **Confidential transactions** that hide amounts, senders, and recipients while preserving on-chain verifiability.
2. **Trust-based credit** that enables multi-currency IOU relationships and cross-currency settlement through an embedded trust-line graph.
3. **Deflationary tokenomics** where every transaction fee is permanently burned and new supply enters only through staking interest—creating a structurally deflationary asset.
4. **Byzantine-fault-tolerant consensus** that provides deterministic finality in seconds without energy-intensive mining.
5. **Decentralised exchange** via an integrated limit-order matching engine operating over the trust-line and native token infrastructure.

The remainder of this paper describes each component in detail, beginning with the cryptographic foundations and proceeding through the ledger model, privacy layer, consensus mechanism, economic design, trust-line infrastructure, and network architecture.

---

2. Cryptographic Foundations

2.1 Elliptic Curve Parameters

NexaFlow operates exclusively over the **secp256k1** elliptic curve [6], the same curve used by Bitcoin [1] and Ethereum [7]. All key generation, digital signatures, Pedersen commitments, ring signatures, and stealth address derivations share this single curve, simplifying the security model to one well-studied assumption: the hardness of the Elliptic Curve Discrete Logarithm Problem (ECDLP) over secp256k1.

The curve is defined over the prime field $\mathbb{F}_p$ with:

$$p = 2^{256} - 2^{32} - 977$$

and group order:

$$n = \texttt{0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141}$$

2.2 Hash Functions

The protocol employs **BLAKE2b-256** [8] as its primary hash function throughout—for transaction identifiers, ledger headers, proposal digests, checksums, and address derivation. BLAKE2b was selected over SHA-256 for its superior throughput on modern hardware and its equivalent or stronger security margins at comparable output lengths.

Two composite hash constructions are used:

- **Double hash:** $H_2(x) = \text{BLAKE2b-256}(\text{BLAKE2b-256}(x))$, used for transaction ID generation and checksum computation.
- **Hash-160:** $H_{160}(x) = \text{RIPEMD-160}(\text{BLAKE2b-256}(x))$, used for address derivation, following the same structural pattern as Bitcoin's address scheme [1] but substituting BLAKE2b for SHA-256 in the first stage.

2.3 Key Generation and Addresses

A wallet generates three independent keypairs over secp256k1:

Keypair	Purpose
Main (k_m, K_m)	General transaction signing
View (k_v, K_v)	Stealth address scanning
Spend (k_s, K_s)	Confidential payment signing and key image generation

Deterministic wallets derive all three from a single seed σ:

$$k_m = H(\sigma), \quad k_v = H(\sigma \| \texttt{"\_view"}), \quad k_s = H(\sigma \| \texttt{"\_spend"})$$

where H denotes BLAKE2b-256 interpreted as an integer modulo n.

An account address is a Base58Check encoding of the Hash-160 of the main public key with a version byte of `0x00`:

$$\text{addr} = \text{Base58Check}(\texttt{0x00} \| H_{160}(K_m))$$

The Base58 alphabet follows the convention established by the XRP Ledger [2]:

`rpshnaf39wBUDNEGHJKLM4PQRST7VWXYZ2bcdeCg65jkm8oFqi1tuvAxyz`

The version byte `0x00` maps to the character `r`, giving all NexaFlow addresses the recognisable `r` prefix.

2.4 Digital Signatures

Standard (non-confidential) transactions are signed with **ECDSA** [9] over secp256k1, producing a DER-encoded signature. The message to be signed is the BLAKE2b-256 digest of the transaction's canonical serialisation (Section 3.2). After signing, the transaction ID is computed as:

$$\text{tx\_id} = H_2(\text{serialize}(tx) \| \text{sig})$$

This binds the transaction identity to both its content and its authorising signature.

3. Transaction Model

3.1 Transaction Types

NexaFlow supports seven transaction types, each identified by an integer code:

Type	Code	Description
Payment	0	Transfer of native NXF or IOU currency
AccountSet	3	Modification of account-level flags
OfferCreate	7	Placement of a limit order on the DEX
OfferCancel	8	Cancellation of an existing DEX order
TrustSet	20	Creation or modification of a trust line
Stake	30	Locking of NXF for staking rewards
Unstake	31	Early cancellation of an active stake

3.2 Serialisation

Each transaction is serialised into a deterministic binary blob for signing. The layout concatenates fields in a fixed order:

```
[4B tx_type] [account] [destination] [amount] [fee]
[8B sequence] [8B timestamp] [type-specific fields] [memo]
[commitment] [stealth_address] [range_proof] [key_image]
[flags as JSON if present]
```

The `ring_signature` field is intentionally excluded from the signing preimage—a signature must never be part of its own input. This is the same structural separation used in Bitcoin's `SIGHASH` mechanism [1].

3.3 Amount Representation

An `Amount` consists of a 64-bit IEEE 754 double-precision value, a 3-byte currency code (padded), and a 40-byte issuer field (empty for native NXF). The native currency is denoted `"NXF"` with no issuer.

3.4 Privacy Fields

When a transaction carries a confidential payment, the following additional fields are populated:

Field	Size	Purpose
`commitment`	65 B	Pedersen commitment hiding the amount
`ring_signature`	Variable	LSAG ring signature proving authorisation

Field	Size	Purpose
`stealth_address`	Variable	One-time recipient address
`ephemeral_pub`	65 B	Sender's ephemeral public key for DH exchange
`view_tag`	1 B	Fast-scan optimisation for recipient
`range_proof`	32 B	Proof that the committed value is non-negative
`key_image`	65 B	Deterministic spend tag for double-spend detection

In a confidential transaction the `amount` field is set to zero; the true value is concealed within the Pedersen commitment.

---

4. Confidential Transactions

NexaFlow's privacy layer draws on techniques originally proposed in CryptoNote [4] and later refined in Monero [3] and Confidential Transactions [10], adapted and integrated into the NexaFlow account model.

4.1 Pedersen Commitments

To hide the value v of a payment, the sender constructs a Pedersen commitment [11]:

$$C = v \cdot G + b \cdot H$$

where G is the secp256k1 base generator, b is a random blinding factor drawn from 32 bytes of `os.urandom`, and H is a second generator whose discrete logarithm with respect to G is unknown. H is derived via a nothing-up-my-sleeve construction:

$$h = \text{SHA-256}(\texttt{"NexaFlow/Pedersen/H-generator/v1"}) \mod n, \quad H = h \cdot G$$

The commitment C is **information-theoretically hiding** (any value is equally likely given C alone) and **computationally binding** (finding two openings (v, b) and (v', b') with $C = v \cdot G + b \cdot H = v' \cdot G + b' \cdot H$ requires solving the ECDLP for H relative to G).

Value precision is maintained at 10^6 decimal places: $v_{\text{int}} = \text{round}(v \times 10^6)$.

Pedersen commitments are additively homomorphic:

$$C_1 + C_2 = (v_1 + v_2) \cdot G + (b_1 + b_2) \cdot H$$

This property enables validators to verify that inputs and outputs balance without knowing the individual values.

4.2 Range Proofs

A commitment alone does not prevent a sender from committing to a negative value, which could be exploited to create supply from nothing. NexaFlow includes a range proof with each confidential output to demonstrate that $v \geq 0$ in zero knowledge. The current implementation uses a BLAKE2b-based deterministic proof keyed on the Pedersen blinding factor and tagged with the domain separator `"NexaFlow/RangeProof/v1"`. The protocol specification reserves this component for replacement with Bulletproofs [12] prior to mainnet launch, which would reduce proof size from $O(n)$ to $O(\log n)$ bits for an n-bit range.

4.3 Stealth Addresses

To prevent linking multiple payments to the same recipient, NexaFlow employs a **Diffie-Hellman stealth address** scheme [4][13]:

Sender protocol:

1. Generate an ephemeral keypair $(r, R = r \cdot G)$.
2. Compute a shared secret using the recipient's view public key: $\text{ss} = \text{SHA-256}((r \cdot K_v).x)$, where $.x$ denotes the x-coordinate of the resulting curve point.
3. Derive the one-time public key: $P = H(\text{ss}) \cdot G + K_s$, where $H(\text{ss}) = \text{int}(\text{ss}) \mod n$.
4. Compute the one-time address: $\text{addr} = \text{derive\_address}(P)$.
5. Compute a 1-byte **view tag**: $\text{vt} = \text{ss}[0]$.
6. Publish $(C, \text{addr}, R, \text{vt})$ on-chain.

Recipient scan:

1. For each unspent output (R, vt, P), compute $\text{ss}' = \text{SHA-256}((k_v \cdot R).x)$.
2. **View tag fast-path:** if $\text{ss}'[0] \neq \text{vt}$, skip immediately. This provides $O(1)$ rejection of approximately $255/256 \approx 99.6\%$ of non-matching outputs [14].
3. Check: $H(\text{ss}') \cdot G + K_s \overset{?}{=} P$.
4. If matched, recover the one-time private key: $k_{\text{ot}} = (H(\text{ss}') + k_s) \mod n$.

Only the holder of the view private key k_v can identify outputs belonging to them, and only the holder of the spend private key k_s can spend them.

4.4 Linkable Ring Signatures (LSAG)

To authorise a confidential payment without revealing which member of a set signed it, NexaFlow uses **Linkable Spontaneous Anonymous Group signatures** [15][16].

Key image generation. Before signing, the signer computes a deterministic tag—the *key image*—that is unique per private key but does not reveal the key itself:

$$I = k_s \cdot H_p(K_s)$$

where H_p is a hash-to-point function: $H_p(P) = (\text{int}(H_{160}(P)) \mod n) \cdot G$.

The key image I serves the same anti-double-spend role as Bitcoin's UTXO consumption [1]: once I appears on the ledger, any subsequent transaction presenting the same I is rejected.

Signing algorithm. Given a ring of m public keys $\{P_0, P_1, \ldots, P_{m-1}\}$ with the real signer at index π:

1. Choose random scalar $k \leftarrow \mathbb{Z}_n$.
2. Compute $L_\pi = k \cdot G$ and $R_\pi = k \cdot H_p(P_\pi)$.
3. Set $c_{(\pi+1) \bmod m} = H(M \| \{P_i\} \| I \| L_\pi \| R_\pi)$, where M is the message (transaction signing hash) and H is BLAKE2b-256.
4. For each subsequent index $i \neq \pi$ (wrapping around the ring), pick random $s_i \leftarrow \mathbb{Z}_n$ and compute:

$$L_i = s_i \cdot G + c_i \cdot P_i$$

$$R_i = s_i \cdot H_p(P_i) + c_i \cdot I$$

$$c_{(i+1) \bmod m} = H(M \| \{P_i\} \| I \| L_i \| R_i)$$

5. Close the ring by setting $s_\pi = (k - c_\pi \cdot k_s) \mod n$.

Verification. A verifier recomputes L_i and R_i for every ring member using the published $(c_0, \{s_i\}, \{P_i\}, I)$ and checks that the challenges form a closed loop: the final computed c matches c_0.

Linkability. Two signatures by the same private key produce the same key image I, enabling double-spend detection without revealing the signer.

4.5 Confidential Payment Flow

A complete confidential payment proceeds as follows:

1. The sender derives a one-time stealth address from the recipient's (K_v, K_s).
2. The sender commits the amount: $C = v \cdot G + b \cdot H$.
3. A range proof is generated proving $v \geq 0$.
4. The sender constructs a ring of public keys—their own spend key K_s at index 0, padded with decoy public keys drawn from the network.
5. The sender signs the transaction's BLAKE2b-256 digest with an LSAG ring signature.
6. The transaction is broadcast with $(C, \text{stealth\_addr}, R, \text{vt}, \text{range\_proof}, \text{ring\_sig}, I)$.
7. The recipient scans unspent outputs using their view key, identifies matching outputs via the view tag, and recovers the one-time spend key.

On-chain, the transaction reveals neither the amount, the sender, nor the recipient—only the Pedersen commitment, the stealth address, and the key image are visible.

---

5. Ledger Model

5.1 Dual-Model Architecture

NexaFlow maintains two parallel state representations within a single ledger:

1. **Account-based state** for standard (transparent) payments, trust lines, offers, and staking. Each account has an address, a native NXF balance, a monotonically increasing sequence number, and a map of trust-line entries.

2. **UTXO-style confidential outputs** for private payments. Each output is a tuple $(\text{commitment}, \text{stealth\_addr}, R, \text{range\_proof}, \text{vt}, \text{tx\_id}, \text{spent})$ indexed by stealth address.

This hybrid approach allows transparent operations to benefit from the simplicity and efficiency of account-based bookkeeping, while confidential payments leverage the privacy guarantees of the UTXO model where each output is independent and unlinkable.

5.2 Double-Spend Prevention

Two independent mechanisms prevent replay and double-spending:

- **Key image set.** For confidential payments, the ledger maintains a set of all spent key images. Before applying a confidential transaction, the key image is checked against this set. Since each private key produces exactly one key image (Section 4.4), this deterministically prevents double-spending without revealing which output was consumed.
- **Applied transaction ID set.** For all transaction types, the ledger maintains a set of applied transaction IDs. Any transaction whose ID has already been recorded is rejected as a duplicate.

5.3 Reserves

To prevent state bloat—a persistent concern in account-based ledger systems [2]—NexaFlow requires each account to maintain a minimum reserve:

$$\text{reserve}(a) = R_{\text{base}} + R_{\text{owner}} \times \text{owner\_count}(a)$$

where $R_{\text{base}} = 20$ NXF and $R_{\text{owner}} = 5$ NXF. The owner count increases by one for each trust line or open offer held by the account. A transaction that would reduce an account's balance below its required reserve is rejected.

5.4 Transaction Fees and Burning

Every transaction pays a minimum fee of 10^{-5} NXF. The fee is deducted from the sender's balance and **permanently destroyed**: it is subtracted from the ledger's `total_supply` and added to a cumulative `total_burned` counter. There is no fee redistribution to validators, miners, or any other party. This mechanism creates persistent deflationary pressure proportional to network activity.

5.5 Ledger Closing and Hash Chaining

At the conclusion of each consensus round (Section 6), the ledger is closed with the following procedure:

1. **Stake maturation.** Any staked positions whose lock period has elapsed are matured: the principal is returned to the staker's account and the accrued interest is newly minted into `total_supply` (Section 7).
2. **Transaction hash.** A Merkle-style digest is computed: $H_{\text{tx}} = \text{BLAKE2b-256}(\text{tx\_id}_1 \| \text{tx\_id}_2 \| \cdots)$.
3. **State hash.** A deterministic snapshot of account state is hashed: $H_{\text{state}} = \text{BLAKE2b-256}(\text{sorted accounts} \| \text{c}$
4. **Header hash.** The ledger header is computed as:

$$H_{\text{header}} = \text{BLAKE2b-256}(\text{seq} \| H_{\text{parent}} \| H_{\text{tx}} \| H_{\text{state}} \| t_{\text{close}} \| N_{\text{tx}} \| S_{\text{total}})$$

where seq is the ledger sequence number, H_{parent} is the hash of the previous closed ledger (or $\texttt{0}^{64}$ for the genesis ledger), t_{close} is the close time, N_{tx} is the transaction count, and S_{total} is the current total supply.

This creates a **hash chain**: each ledger header cryptographically commits to the entire history of the network, providing tamper evidence analogous to Bitcoin's block chain [1].

5.6 Confidential Payment Application

A confidential transaction undergoes a six-step verification and application pipeline:

1. **Duplicate check** — reject if `tx_id` is already in the applied set.
2. **Range proof verification** — confirm the committed value is non-negative.
3. **Ring signature verification** — confirm the LSAG signature is valid over the transaction's signing hash.
4. **Fee deduction and burn** — debit the fee from the sender's public account balance; destroy it.
5. **UTXO creation** — store the `ConfidentialOutput` indexed by stealth address.
6. **Spend recording** — add the key image to the spent set and the transaction ID to the applied set.

5.7 Transaction Validation Pipeline

Before any transaction reaches the application layer, it passes through a multi-stage validation pipeline:

1. **Signature verification** — ECDSA signature must be valid for the signing public key.
2. **Privacy checks** (if confidential fields are present): key image freshness, range proof validity, ring signature validity.
3. **Account existence** — the source account must exist in the ledger.
4. **Fee adequacy** — fee $\geq 10^{-5}$ NXF.
5. **Sequence correctness** — if non-zero, must match the account's current sequence.
6. **Balance and reserve adequacy** — the account must retain at least its required reserve after the transaction.

---

6. Consensus

6.1 Overview

NexaFlow reaches agreement through a **Byzantine-Fault-Tolerant variant of the Ripple Protocol Consensus Algorithm (BFT-RPCA)** [2][17]. Unlike proof-of-work [1] or proof-of-stake [18] mechanisms, RPCA achieves deterministic finality within a bounded number of voting rounds without requiring energy-intensive computation or stake-weighted leader election.

6.2 Unique Node List

Each validator maintains a **Unique Node List (UNL)**—a set of validators it trusts for consensus purposes. The UNL is analogous to a quorum slice in Federated Byzantine Agreement [19], but with the additional constraint that validators sign their proposals with secp256k1 ECDSA, enabling cryptographic verification of every vote.

6.3 Consensus Phases

The consensus process moves through four phases:

Phase	Name	Description
0	Open	Accepting new transactions into the candidate set
1	Establish	Iterative voting rounds in progress
2	Accepted	Consensus reached; ledger closing
3	Failed	No agreement after maximum rounds

6.4 Proposal Structure

Each proposal contains the validator's identity, the target ledger sequence, a set of candidate transaction IDs, a timestamp, a round number, and a DER-encoded ECDSA signature. The proposal hash is computed as:

$$H_{\text{prop}} = \text{BLAKE2b-256}(\text{validator\_id} \| \text{ledger\_seq} \| \text{round} \| \text{sorted\_tx\_ids})$$

6.5 Threshold Escalation

Voting proceeds over a maximum of 10 rounds. The support threshold—the fraction of UNL members that must include a transaction for it to remain in the candidate set—escalates linearly from an initial threshold to a final threshold:

$$\theta(r) = \min\left(\theta_0 + \frac{\theta_f - \theta_0}{\text{max\_rounds} - 1} \times r,\ \theta_f\right)$$

where $\theta_0 = 0.50$ (50%), $\theta_f = 0.80$ (80%), and r is the round index starting from 0.

A transaction survives round r if and only if:

$$\frac{\text{votes}(tx)}{\text{honest\_validators}} \geq \theta(r)$$

The final threshold of 80% exceeds the 2/3 supermajority required by classical BFT results [20], providing safety under the standard assumption:

$$n \geq 3f + 1, \quad f = \left\lfloor \frac{n-1}{3} \right\rfloor$$

where n is the total number of UNL members (including self) and f is the maximum number of Byzantine faults tolerated.

6.6 Equivocation Detection

If a validator submits two different proposals for the same ledger sequence and round—an equivocation—both proposals are discarded and the validator is permanently flagged as Byzantine. Flagged validators are excluded from the quorum denominator, ensuring that their conflicting votes cannot deadlock or corrupt the consensus process.

6.7 Convergence

Between rounds, each honest validator updates its own proposal to the current agreed set (the intersection of transactions that survived the threshold). This convergence behaviour causes proposals to align over successive rounds, accelerating agreement.

---

7. Tokenomics and Staking

7.1 Native Token

The native currency of NexaFlow is **NXF**. At genesis, 100 billion (10^{11}) NXF are minted to the genesis account. There is no block reward, no inflation schedule, and no administrative minting capability. After genesis, new NXF enters circulation only through staking interest (Section 7.3), and NXF is permanently removed from circulation only through fee burning (Section 5.4) and early-cancellation principal penalties (Section 7.5).

7.2 Deflationary Dynamics

The token supply $S(t)$ at time t evolves according to:

$$S(t) = S_0 - B(t) + M(t)$$

where $S_0 = 10^{11}$ is the initial supply, $B(t)$ is the cumulative fees and principal penalties burned, and $M(t)$ is the cumulative interest minted. Under normal network activity the burn rate is expected to exceed interest minting, making NXF structurally deflationary.

7.3 Tiered Staking

NexaFlow supports five staking tiers with increasing lock durations and base annual percentage yields:

Tier	Lock Duration	Base APY
0 — Flexible	None	2 %
1 — 30 Days	30 days	5 %
2 — 90 Days	90 days	8 %
3 — 180 Days	180 days	12 %
4 — 365 Days	365 days	15 %

A stake is recorded on-chain as a Stake transaction (type 30). The stake ID equals the transaction ID, guaranteeing a one-to-one correspondence and preventing duplication. The minimum stake amount is 1.0 NXF.

7.4 Dynamic Interest Rate

The effective APY for any tier is the product of its base APY and a **demand multiplier** that adjusts based on the network's staking ratio:

$$\mu = \mathrm{clamp}(0.5,\ 2.0,\ 1 + (T - r) \times \kappa)$$

$$\mathrm{APY}_{\mathrm{eff}} = \mathrm{APY}_{\mathrm{base}} \times \mu$$

where: - $T = 0.30$ is the target staking ratio (30 % of circulating supply), - $r = S_{\mathrm{staked}}/S_{\mathrm{circulating}}$ is the actual staking ratio, - $\kappa = 3.0$ is the demand sensitivity coefficient, - μ is clamped to the range $[0.5, 2.0]$.

When fewer tokens are staked ($r < T$), the multiplier rises above 1, incentivising capital to enter staking pools. When too many are staked ($r > T$), the multiplier falls below 1, releasing liquidity back to circulation. This creates a negative feedback loop that self-regulates around the target ratio.

7.5 Interest Accrual and Maturation

Interest accrues proportionally over the lock period. At maturity (checked during each ledger close), the principal is returned to the staker's account and the interest is newly minted into `total_supply`:

$$I = P \times \mathrm{APY}_{\mathrm{eff}} \times \frac{d}{Y}$$

where P is the principal, d is the lock duration in seconds, and Y = 31,557,600 seconds (365.25 days).

7.6 Early Cancellation

A locked-tier stake may be cancelled before maturity by submitting an Unstake transaction (type 31). Early cancellation incurs two penalties that scale with both the tier's base APY and the remaining lock time:

Interest penalty rate:

$$\rho_I = 0.50 + \frac{\mathrm{APY}_{\mathrm{base}}}{0.15} \times 0.40$$

Principal penalty rate:

$$\rho_P = 0.02 + \frac{\mathrm{APY}_{\mathrm{base}}}{0.15} \times 0.08$$

Time decay factor:

$$\delta = 1 - \frac{t_{\text{elapsed}}}{d_{\text{lock}}}$$

where t_{elapsed} is the time since staking and d_{lock} is the total lock duration. The decay factor linearly reduces the penalties as the stake approaches maturity, reaching zero at maturity.

Final payout:

$$\text{payout} = (P - P \cdot \rho_P \cdot \delta) + (I_{\text{accrued}} - I_{\text{accrued}} \cdot \rho_I \cdot \delta)$$

The principal penalty is permanently burned, creating additional deflationary pressure. Flexible-tier stakes ($d_{\text{lock}} = 0$) always have $\delta = 0$, meaning zero penalty.

---

8. Trust Lines and Cross-Currency Settlement

8.1 Trust Line Model

Building on the credit network concepts introduced in Ripple [5][2], NexaFlow supports **trust lines** between accounts. A trust line is a directed credit relationship in which a *holder* agrees to accept IOUs denominated in a specific currency from an *issuer*, up to a configurable credit limit.

A `TrustLineEntry` stores: the currency code, the issuer, the holder, the current balance, the holder's limit, an optional counter-limit set by the issuer, and a `no_ripple` flag that prevents the account from being used as an involuntary intermediary.

Trust lines are created or modified via TrustSet transactions (type 20).

8.2 Trust Graph

The set of all trust lines forms a directed graph—the **trust graph**—with edges weighted by available credit:

$$\text{credit}(h, i, c) = \max(0,\ \text{limit}(h, i, c) - \text{balance}(h, i, c))$$

where h is the holder, i is the issuer, and c is the currency. This graph is the foundation for cross-currency payment path finding and rippling.

8.3 Payment Path Finding

When a payment cannot be settled directly (e.g., the sender and recipient do not share a trust line in the desired currency), NexaFlow searches the trust graph for multi-hop settlement paths.

The path-finding algorithm is a **depth-first search with backtracking**:

1. Start at the source account.
2. At each node, check whether the destination has a trust line to the current node for the target currency. If so, record the available credit as a candidate path.

3. Otherwise, explore all accounts that trust the current node as an issuer (reverse adjacency in the trust graph).
4. Maintain a visited set to prevent cycles; backtrack to discover multiple paths.
5. Terminate after discovering 5 paths or exploring 6 hops, whichever comes first.

Discovered paths are ranked by $(-\text{max\_amount}, \text{hop\_count})$: highest liquidity first, fewest hops as tiebreaker.

8.4 Rippling

Settlement along a discovered path operates through **rippling**: at each hop, the sender's trust-line balance toward the intermediary is debited, and the intermediary's trust-line balance toward the next hop is credited. This enables payment across chains of bilateral trust without any single party needing to hold every currency.

---

9. Decentralised Exchange

9.1 Order Book

NexaFlow includes an integrated **limit-order matching engine** that operates over the trust-line and native token infrastructure. An order specifies a trading pair (e.g., `NXF/USD`), a side (buy or sell), a price, and a quantity. Orders are submitted via OfferCreate transactions (type 7) and cancelled via OfferCancel transactions (type 8).

9.2 Price-Time Priority

Orders are sorted by strict **price-time priority**:

$$\text{sort}(\text{ask}) = (+\text{price}, +\text{timestamp}), \quad \text{sort}(\text{bid}) = (-\text{price}, +\text{timestamp})$$

Ask orders are displayed lowest-price-first; bid orders highest-price-first. Equal-price orders are resolved by arrival time. Insertion uses binary search ($O(\log n)$) to maintain sorted order.

9.3 Matching Algorithm

When a new order arrives:

1. **Buy orders** are matched against resting asks, starting from the lowest ask price.
2. **Sell orders** are matched against resting bids, starting from the highest bid price.
3. Matching stops when no resting order satisfies the taker's price limit.
4. The fill quantity at each step is $\min(\text{taker.remaining}, \text{maker.remaining})$.
5. If the taker has remaining quantity after exhausting all matchable resting orders, the unfilled portion rests on the book.

All fills are recorded with the maker's price (price improvement for the taker), the fill quantity, and a timestamp.

---

10. Network Architecture

10.1 Peer-to-Peer Transport

NexaFlow nodes communicate over **TCP** using newline-delimited JSON messages. Each message contains a `type` field, a `payload` object, and a `ts` timestamp. The protocol optionally supports **TLS 1.3** for transport encryption and **mutual TLS (mTLS)** for validator identity verification [21].

10.2 Message Types

Type	Purpose
`HELLO`	Handshake with node identity and optional public key
`TX`	Broadcast a signed transaction
`PROPOSAL`	Consensus proposal for a ledger round
`CONSENSUS_OK`	Agreed transaction set after consensus
`LEDGER_REQ` / `LEDGER_RES`	Ledger state synchronisation
`PING` / `PONG`	Keepalive (60-second timeout)

10.3 Peer Discovery

On startup, a node connects to a configured set of seed peers. Upon successful connection, each peer exchanges a `HELLO` message containing its node ID and, optionally, its secp256k1 public key for identification. New transactions and consensus proposals are broadcast to all connected peers.

10.4 REST API

Each node optionally exposes an `aiohttp`-based HTTP API for external integration. POST endpoints are protected by API-key authentication, per-IP token-bucket rate limiting (configurable, default 120 requests per minute), CORS middleware, and a request body size cap (default 1 MiB).

---

11. Wallet Encryption

Private keys at rest are protected using a two-layer encryption scheme:

1. **Key derivation:** PBKDF2-HMAC-SHA256 with 100,000 iterations and a 16-byte random salt, deriving a 32-byte encryption key from the user's passphrase.
2. **Stream cipher:** A BLAKE2b-CTR construction: for each 32-byte block, a keystream block is generated as BLAKE2b-256(key‖counter) where the counter is a 128-bit value incremented per block from a random IV.

This avoids any dependency on external encryption libraries while providing semantic security under a strong passphrase. The encrypted wallet format (version 2) stores all three private keys (main, view, spend), the salt, and the IV.

---

12. Performance

All performance-critical cryptographic operations—Pedersen commitments, ring signature generation and verification, stealth address derivation, range proof construction, transaction serialisation, and ledger hashing—are implemented in **Cython** [22] and compiled to native C extension modules. This eliminates Python interpreter overhead for the hot path while retaining the development velocity and ecosystem of Python for the application layer, networking, and API surfaces.

---

13. Security Considerations

- All cryptographic operations use the **secp256k1** curve, the same curve securing hundreds of billions of dollars on the Bitcoin and Ethereum networks [1][7].
- The Pedersen commitment generator H is derived via a deterministic nothing-up-my-sleeve construction, ensuring no party knows its discrete logarithm relative to G.
- Ring signatures provide **k-anonymity** for the sender within a ring of size k—an adversary who observes a transaction cannot distinguish the true signer from $k - 1$ decoys.
- Stealth addresses provide **unlinkability** for the recipient—an adversary who observes multiple payments to the same recipient sees only distinct one-time addresses, each indistinguishable from random.
- The BFT-RPCA consensus guarantees safety as long as fewer than one-third of UNL members are Byzantine, aligning with the classical BFT bound [20].
- Wallet private keys are never stored in plaintext—the PBKDF2 + BLAKE2b-CTR encryption scheme provides semantic security under the assumption that the user's passphrase has sufficient entropy.
- The protocol enforces input validation and reserve requirements on all public API surfaces to prevent state-bloat attacks and resource exhaustion.

Disclosure. While the cryptographic design follows established constructions from peer-reviewed literature, the NexaFlow codebase has not yet undergone a formal third-party security audit. Users should exercise appropriate caution until an independent audit is completed.

---

14. Conclusion

NexaFlow demonstrates that confidential transactions, trust-based credit networks, deflationary tokenomics, and Byzantine-fault-tolerant consensus can coexist within a single protocol without fundamental compromise. By layering Pedersen commitments, LSAG ring signatures, and stealth addresses over a hybrid account-and-UTXO ledger model, the network simultaneously supports fully transparent operations for use cases that require auditability and fully private payments for use cases that demand confidentiality. The dynamic staking mechanism self-regulates the balance between supply contraction (fee burning) and supply expansion (interest minting), and the integrated trust-line graph and decentralised exchange enable multi-currency settlement without external bridges. We believe this unified architecture addresses a practical gap in the design space of digital payment networks.

References

[1] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008. Available: https://bitcoin.org/bitcoin.pdf

[2] D. Schwartz, N. Youngs, and A. Britto, “The Ripple Protocol Consensus Algorithm,” Ripple Labs, 2014. Available: https://ripple.com/files/ripple_consensus_whitepaper.pdf

[3] S. Noether, “Ring Confidential Transactions,” *Ledger*, vol. 1, pp. 1–18, 2016. doi: 10.5195/ledger.2016.34

[4] N. van Saberhagen, “CryptoNote v 2.0,” 2013. Available: https://cryptonote.org/whitepaper.pdf

[5] R. Fugger, “Money as IOUs in Social Trust Networks & A Proposal for a Decentralized Currency Network Protocol,” 2004. Available: http://ripple.ryanfugger.com/

[6] Certicom Research, “SEC 2: Recommended Elliptic Curve Domain Parameters,” Standards for Efficient Cryptography Group, Version 2.0, 2010. Available: https://www.secg.org/sec2-v2.pdf

[7] V. Buterin, “Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform,” 2014. Available: https://ethereum.org/en/whitepaper/

[8] J.-P. Aumasson, S. Neves, Z. Wilcox-O’Hearn, and C. Winnerlein, “BLAKE2: simpler, smaller, fast as MD5,” in *Applied Cryptography and Network Security (ACNS)*, Springer, 2013, pp. 119–135. doi: 10.1007/978-3-642-38980-1_8

[9] D. Johnson, A. Menezes, and S. Vanstone, “The Elliptic Curve Digital Signature Algorithm (ECDSA),” *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63, 2001. doi: 10.1007/s102070100002

[10] G. Maxwell, “Confidential Transactions,” 2015. Available: https://elementsproject.org/features/confidential-transactions

[11] T. P. Pedersen, “Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing,” in *Advances in Cryptology — CRYPTO ’91*, Springer, 1992, pp. 129–140. doi: 10.1007/3-540-46766-1_9

[12] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, “Bulletproofs: Short Proofs for Confidential Transactions and More,” in *IEEE Symposium on Security and Privacy*, 2018, pp. 315–334. doi: 10.1109/SP.2018.00020

[13] P. Todd, “Stealth Addresses,” Bitcoin Development Mailing List, 2014. Available: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2014-January/004020.html

[14] “View Tags for Monero Stealth Addresses,” Monero Research Lab, MRL-0011, 2022.

[15] J. K. Liu, V. K. Wei, and D. S. Wong, “Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups,” in *ACISP 2004*, Springer, 2004, pp. 325–335. doi: 10.1007/978-3-540-27800-9_28

[16] A. Back, “Ring Signature Efficiency,” Blockstream Research, 2015.

[17] B. Chase and E. MacBrough, “Analysis of the XRP Ledger Consensus Protocol,” *arXiv preprint arXiv:1802.07242*, 2018. Available: https://arxiv.org/abs/1802.07242

[18] S. King and S. Nadal, “PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake,” 2012. Available: https://peercoin.net/whitepapers/peercoin-paper.pdf

[19] D. Mazières, "The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus," Stellar Development Foundation, 2015. Available: https://www.stellar.org/papers/stellar-consensus-protocol.pdf

[20] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI)*, 1999, pp. 173–186.

[21] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Internet Engineering Task Force, 2018. doi: 10.17487/RFC8446

[22] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, "Cython: The Best of Both Worlds," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 31–39, 2011. doi: 10.1109/MCSE.2010.118