

# NITO WHITEPAPER

## Proof of Work, Monetary Policy and Optional Privacy

nito.network

# Overview

Nito is a peer-to-peer payment network with a native currency, NITO. It combines SHA-256 proof of work, a target block interval of one minute and an issuance schedule defined by block height. Users hold their own keys. Miners order transactions, while full nodes independently apply the rules that determine which payments and blocks are valid.

The ledger supports transparent payments and optional private payments. Transparent outputs publish their values and spending conditions. Private payments use Orchard notes, encrypted payment data and Halo 2 zero-knowledge proofs. A node can verify authorization and conservation of value without learning the recipients or individual values of private notes.

Both payment forms use the same currency and blockchain. Shielding moves existing NITO into private notes; unshielding moves value back into transparent outputs. Neither operation creates a second asset. The mining algorithm and 80-byte block header serve the whole ledger.

Nito Core v4 defines mainnet Orchard activation at **block 1,165,432**. This is the first height at which a block may contain an Orchard transaction. The private payment rules described here apply from that height. Transparent payments remain available, and Bech32 remains the default receiving format.

## Reading Guide

| Sections | Subject |
|---|---|
| 1-2 | The blockchain, transparent payments and addresses |
| 3-5 | Proof of work, issuance, mining and transaction fees |
| 6-9 | Orchard, value conservation and transaction validation |
| 10 | Information visible on the ledger and privacy limits |
| 11-13 | Keys, backups, recovery and blockchain storage |
| 14-15 | Activation, network operation and security assumptions |
| 16-17 | Glossary and technical references |

Amounts in formulas are exact integer amounts unless a worked example explicitly uses NITO. The smallest unit is one nitoshi, equal to one hundred-millionth of one NITO. Mainnet parameters are stated explicitly; test networks have separate chains and settings.

The protocol description follows the Nito Core implementation identified in Reference 1. The references also distinguish Nito rules from the Bitcoin and Zcash constructions on which parts of the design are based.

# 1. Blockchain Architecture

## 1.1 Nodes, Miners and Wallets

A **full node** receives transactions and blocks from peers and checks them against its local rules and chainstate. A **miner** assembles a candidate block and searches for valid proof of work. A **wallet** manages keys, discovers payments and constructs transactions. These roles can run together, but they serve different purposes: a valid proof of work does not authorize an otherwise invalid payment.

Blocks refer to their predecessors by hash. Within each block, a Merkle root commits to the ordered transaction identifiers. Changing a transaction changes this commitment and consequently the header hash. A separate witness commitment covers the transaction authorization data carried outside the stripped transaction representation.

## 1.2 Mainnet Parameters

| Parameter | Value |
|---|---|
| Currency and unit | NITO; 100,000,000 nitoshi per NITO |
| Proof of work | SHA-256d: two successive SHA-256 hashes |
| Target block interval | 60 seconds |
| Retarget frequency | Every block |
| Header length | 80 bytes |
| Maximum block weight | 800,000 weight units |
| Serialized block-size bound | 800,000 bytes |
| Maximum block signature-operation cost | 80,000 |
| Coinbase maturity | 100 blocks |
| Maximum spendable scheduled issuance | 1,284,565,890 NITO |

Block weight is the binding capacity measure described in Section 5. The serialized-size bound is not an allowance for 800,000 bytes of arbitrary non-witness data.

## 1.3 Two Forms of Ledger State

Transparent state is the set of **unspent transaction outputs**, or UTXOs. Spending removes available outputs and creates new ones. Orchard state consists of a note commitment tree, accepted tree roots, spent nullifiers and an aggregate private-pool balance. A node maintains both forms of state as part of one chain.

State updates follow block order. If the active chain changes, the node reverses the disconnected transactions before applying the replacement branch. Private tree state, nullifiers and pool accounting must change together with transparent UTXOs.

## 1.4 Confirmations

Inclusion in a block gives a transaction its first confirmation. Subsequent blocks add accumulated work above it. Nodes select a valid chain by accumulated proof of work, not simply by counting blocks. Settlement remains probabilistic: the one-minute target is neither a delivery deadline nor a guarantee of irreversible payment.

# 2. Transparent Payments

## 2.1 Outputs and Spending Conditions

A transparent output contains an integer value and a script specifying how it may be spent. An input identifies an earlier output by transaction identifier and output index, then provides the required authorization. Nodes reject missing or already-spent inputs, duplicated inputs, invalid scripts and out-of-range amounts.

For an ordinary transaction with no Orchard bundle, total input value must cover total output value. The difference is the fee. Change is a new output controlled by the sender; the original output is consumed in full. A wallet balance is therefore a view of spendable outputs, not an account balance stored by the network.

## 2.2 Address Families

| Address family | Mainnet prefix | Use |
|---|---|---|
| Legacy P2PKH | 1... | Transparent public-key-hash payment |
| P2SH | 3... | Transparent script-hash payment, including wrapped SegWit |
| Native SegWit, Bech32 | nito1q... | Transparent; default receive format |
| Taproot, Bech32m | nito1p... | Transparent key-path or script-path payment |
| Orchard, Bech32x | nito1x... | Private receiver selected explicitly |

An address encodes a payment destination; it is not itself a stored balance or a private key. Checksums help detect transcription errors but do not establish the identity of the recipient. The sender must select the correct network and payment type.

## 2.3 Signatures and Scripts

Legacy and SegWit version-0 spending use ECDSA signatures over secp256k1 where signatures are required. Taproot uses Schnorr signatures over the same curve and supports a direct key path or disclosure of a selected script path. Script rules support conditions such as multiple signatures and time locks. They are a constrained validation language, not a general-purpose application execution environment.

SegWit separates witness data from the stripped transaction representation. Taproot can avoid revealing unused alternative spending conditions, but it does not conceal the value or destination of a transparent output. Neither format provides the ledger privacy of an Orchard note.

## 2.4 Maturity and Time Locks

Newly mined coinbase outputs require a difference of at least 100 between their creation height and spending height. This protects recipients of mining rewards against spending rewards from very recent blocks that may be reorganized.

Absolute lock time can delay eligibility until a block height or time threshold. Relative sequence locks can require age since an input was confirmed. Applicable time-based checks use the median-time rules of the chain, rather than a guarantee based on a user's wall clock. A valid signature does not bypass maturity or lock-time requirements.

# 3. Proof of Work and Difficulty

## 3.1 Header and Work

The 80-byte header contains version (4 bytes), previous block hash (32), transaction Merkle root (32), timestamp (4), compact target (4) and nonce (4). A valid candidate satisfies the following comparison, interpreting the hash as an unsigned 256-bit integer:

```
SHA256(SHA256(header)) <= T
```

Here, **T** is the target decoded from the compact target field. Zero, negative, overflowing and above-limit targets are invalid. The mainnet proof-of-work limit is **2^224 - 1**. A smaller target makes a successful hash less likely.

Under the uniform-hash model, a trial succeeds with probability **(T + 1) / 2^256**. The expected number of trials is its reciprocal. The integer work credited to a block is:

```
work(T) = floor(2^256 / (T + 1))
```

A chain's work is the sum of these contributions. This measures the work implied by its targets; it is not a measurement of the actual number of hashes miners attempted.

## 3.2 Per-Block Retargeting

Let **dt** be the difference in seconds between the timestamps of the two preceding blocks. For the first retarget after genesis, there is no preceding pair and the measured interval is zero. The algorithm first dampens the observed interval:

```
S = 60 + trunc((dt - 60) / 8)
```

Here, **trunc** is integer division toward zero, including when its argument is negative. The interval is then bounded and applied to the previous decoded target:

```
S = min(90, max(45, S))

T_next = min(2^224 - 1, floor(T_previous * S / 60))
```

The result is encoded back into compact form, which introduces precision loss. Before compact rounding and the target cap, one adjustment multiplies the target by between 0.75 and 1.5. Difficulty varies inversely with target. For example, an observed interval of 140 seconds gives S = 70 and a target multiplier of 7/6, before rounding and capping.

## 3.3 Time and Hash-Rate Changes

A header timestamp must exceed the median of the previous 11 block timestamps, or the available preceding timestamps near genesis. A node also rejects a header more than two hours ahead of its current clock until it becomes acceptable. Consecutive header timestamps need not increase individually.

Damping limits the response to one observed interval. It does not prevent long waits after a large loss of hash rate, nor does it make miner-supplied timestamps exact measures of elapsed real time. Mainnet has no delayed-block exception that immediately grants minimum difficulty.

# 4. Monetary Policy

## 4.1 Height-Based Issuance

The block subsidy is the maximum new currency a miner may claim at a particular height. The schedule uses explicit height ranges; it is not a uniform repeating halving interval. All endpoints below are inclusive.

| Block heights | Subsidy per block | Scheduled NITO in range |
|---|---|---|
| 0-529,999 | 512 NITO | 271,360,000 |
| 530,000-1,042,399 | 256 NITO | 131,174,400 |
| 1,042,400-1,576,799 | 128 NITO | 68,403,200 |
| 1,576,800-5,255,999 | 64 NITO | 235,468,800 |
| 5,256,000-10,511,999 | 32 NITO | 168,192,000 |
| 10,512,000-26,279,999 | 16 NITO | 252,288,000 |
| 26,280,000-105,120,000 | 2 NITO | 157,680,002 |
| 105,120,001 onward | 0 NITO | 0 |

For a range from height **a** through **b**, with subsidy **r**, its scheduled amount is **(b - a + 1) * r**. The final 2-NITO range includes block 105,120,000. Summing all ranges, including genesis, gives **1,284,566,402 NITO**.

## 4.2 Supply Bound

The genesis coinbase contains 512 NITO, but its output is not inserted into spendable chainstate. The maximum scheduled issuance that can enter the spendable ledger is therefore:

> **1,284,566,402 - 512 = 1,284,565,890 NITO**

At height **h**, let **R(k)** be the scheduled subsidy at height k. The cumulative spendable issuance ceiling is:

> **S_max(h) = sum of R(k), for k = 1 through h**

This is a protocol ceiling, not circulating supply measured in markets. A miner can claim less than the allowed reward. Provably unspendable outputs and lost keys further reduce the value available to holders. Fees transfer existing coins and must not be counted again as new issuance.

The implementation also uses **1,284,565,890 NITO** as its maximum permitted money amount in range checks. That check alone does not prove conservation: the issuance bound also depends on subsidy validation, transaction accounting and the correctness of private proofs and signatures.

## 4.3 Duration and Mining Revenue

At the target interval, 105,120,000 post-genesis blocks represent approximately 200 years using 365-day years. The schedule is enforced by height, not by a predicted calendar date. Actual block production varies with hash rate, difficulty and chance.

The subsidy eventually reaches zero. Under these rules, mining revenue then comes from transaction fees. The protocol does not provide a perpetual tail emission or a mandatory treasury share. Future fee demand and the resulting level of mining security are economic outcomes, not guarantees of the issuance formula.

# 5. Mining, Capacity and Fees

## 5.1 Block Rewards

The first transaction in each block is the coinbase. Its total output value may not exceed that height's subsidy plus the fees of the other transactions. Coinbase transactions are transparent and cannot contain an Orchard bundle. Private payments introduce neither a separate mining reward nor a required payment to a privacy service.

SHA-256 mining hardware hashes the same header format for transparent and private transactions. A pool must upgrade its backing node to Nito Core v4 before activation. That node validates Orchard proofs and supplies valid transactions through **getblocktemplate** (GBT). Pool software that already handles the standard GBT and SegWit requirements below can operate without an Orchard-specific modification. It does not need to generate or independently verify the payment proofs supplied by senders.

## 5.2 Pool Block Construction

**Complete transactions.** The pool requests GBT with **rules: ["segwit"]** and copies each selected entry's **data** bytes unchanged into the submitted block. Those bytes include Orchard effects, proofs and signatures. Treating them as opaque data avoids the need for an Orchard-aware transaction parser. A parser that rejects, strips or rewrites unfamiliar fields is not sufficient.

**Transaction identifiers.** The header's Merkle root uses the ordered **txid** values, preceded by the actual coinbase txid. The separate GBT **hash** field is the witness transaction identifier, or wtxid. It includes authorization data and is used for the witness commitment. Hashing the complete **data** bytes is not a substitute for the stripped **txid**.

**Witness commitment.** For an unchanged transaction list and order, the pool can include Core's **default_witness_commitment** script in a zero-value coinbase output. The coinbase input witness must contain exactly one 32-byte item; Core's default commitment uses 32 zero bytes. The witness Merkle tree uses the transactions' wtxids with a zero-valued coinbase leaf. Its commitment is:

```
commitment = SHA256d(witness_root || witness_reserved_value)
```

Here **||** denotes byte concatenation. The output script is the prefix **6a24aa21a9ed** followed by the 32-byte commitment. Changing the coinbase extra nonce changes its txid and requires a new header Merkle root, but does not change the zero-valued witness-tree leaf.

**Selection and fees.** Removing or reordering transactions requires preserving parent-before-child dependencies and rebuilding the affected Merkle roots and witness commitment. The **depends** field identifies parents by their 1-based positions in the template's transaction array. Core's **coinbasevalue** is the maximum reward for the complete template. For a reduced selection, the pool must deduct omitted transaction fees, using the GBT **fee** amounts in nitoshis. Transparent inputs and outputs alone do not determine an Orchard transaction's fee; Section 7 gives the full value equation. The final block, including the pool's coinbase, must respect the template's **weightlimit**, **sigoplimit** and **sizelimit**.

## 5.3 Pool Verification and Payouts

An operator can check block construction on testnet or regtest with templates containing shielding, private transfers and unshielding. GBT's **mode: "proposal"** accepts the complete candidate block in **data**. For a new candidate extending the node's current tip, a **null** result means the validation performed by this mode succeeded; proof of work is not checked. This includes transaction validity, coinbase value and transaction/witness commitments. A solved block is then submitted in full with **submitblock**; a newly accepted block returns **null**. Rejection or an inconclusive response must not be treated as acceptance.

Coinbase rewards remain transparent. Paying miners later to Bech32x addresses (**nito1x...**) is an optional payout feature, supported through the Core wallet. A pool can include Orchard transactions while continuing to pay every miner to a transparent address. Supporting private payout addresses is not a condition for mining Orchard-containing blocks.

The mining interface and pool compatibility requirements provide the detailed integration reference.

## 5.4 Weight and Virtual Size

Let **B** be stripped size in bytes and **F** be full serialized size. Stripped data includes the Orchard effects, while full data additionally includes applicable transparent witness and private authorization data. Weight is:

$$W = 3 * B + F$$

Equivalently, each stripped byte contributes four weight units and each additional byte contributes one. The block weight limit is 800,000. A block with entirely non-witness data is consequently limited to approximately 200,000 bytes, including block overhead.

For fee and relay policy, Nito Core also accounts for signature operations and Orchard verification. Let **C** be the signature-operation cost and **b** the configured bytes-per-signature-operation setting, whose default is 20. For **n** Orchard actions:

$$V_{serial} = \text{ceil}(\max(W, C * b) / 4)$$

$$V_{private} = 5{,}000 + 1{,}000 * n$$

$$V_{policy} = \max(V_{serial}, V_{private})$$

For a transaction without Orchard, V_private is zero. For example, a two-action bundle has a private verification floor of 7,000 virtual bytes; a larger serialized or signature-operation cost still takes precedence. These are policy units, not a claim that the transaction occupies that many bytes on disk.

## 5.5 Relay and Fee Selection

The fee rate is the transaction fee divided by its policy virtual size. Relay thresholds, dust rules, mempool limits and replacement rules are local policies. They affect propagation and miner selection; they do not all define block validity. A transaction accepted by one node is not guaranteed immediate inclusion by a miner.

Private conflicts are detected through nullifiers as well as transparent inputs. Unconfirmed transactions may be evicted, replaced under applicable policy or invalidated by a competing confirmed spend. A mempool entry is not a settled payment. The fee is public even when both payment endpoints are private, as the conservation equation in Section 7 shows.

# 6. Orchard Cryptography

## 6.1 Notes, Commitments and Anchors

An Orchard **note** assigns a value to a recipient and includes cryptographic randomness. A commitment binds these contents while concealing them. New commitments are appended to a depth-32 Merkle tree, with a capacity of **2^32 commitments**. A spender proves membership under an accepted root, or **anchor**, without publishing the consumed note's tree position.

A **nullifier** identifies a spend without identifying its original note commitment to the public. Its derivation is tied to the note and appropriate key material. Nodes reject a nullifier already used in the active chain, preventing a note from being spent twice. These properties rely on the Orchard construction and its cryptographic assumptions.

## 6.2 Actions and Encrypted Outputs

An Orchard action combines a spend side and an output side. Bundles can include zero-value dummy notes, so the visible action count is not necessarily the number of recipients or real notes spent. A Nito bundle contains between **1 and 50 actions**. Flags indicate whether spends and outputs are enabled.

Each action publishes a net value commitment, a nullifier, a randomized authorization key, a note commitment, an ephemeral encryption key and ciphertexts. Recipient encryption enables wallet discovery without publishing the private address or amount. Outgoing viewing data can allow the sender to recover its own payment details.

## 6.3 Proofs and Signatures

Halo 2 proves that the hidden note data satisfies the Orchard circuit: applicable input membership, valid note construction, correct nullifier derivation and the required relationships between keys, commitments and values. The construction uses the Pasta elliptic curves and an inner-product polynomial commitment; it does not require a trusted setup ceremony. This removes setup-secret trust, not the need for sound cryptography and correct software. [3, 4]

**RedPallas spend-authorization signatures** authorize actions under their randomized keys. A **binding signature** links the aggregate value commitments to the declared public value balance. Proof verification, spend authorization and binding verification must all succeed. A proof alone is not permission to spend somebody else's notes.

For Nito Core v4, the verifier selects Orchard V2 bundle semantics and the fixed post-NU6.2 circuit, identified by **FixedPostNu6_2** in the implementation. These cryptographic versions are distinct from the Nito private serialization version, which is 1. A change to the accepted circuit is a protocol decision, not an automatic consequence of updating a dependency. [1]

## 6.4 A Payment Construction

The sender selects notes or transparent inputs, creates recipient notes and change, encrypts the note data, constructs the proof and signs the transaction. Peers validate the public data and authorization. After confirmation, recipients scan the encrypted outputs using their viewing authority and maintain the note data needed for a future spend.

Public observers can validate the state transition without obtaining this wallet data. They can still see the transaction and its public fields; Section 10 describes those limits.

# 7. Conservation of Value

## 7.1 One Currency Across Both Payment Forms

Let **T_in** be the sum of transparent input values, **T_out** the sum of transparent output values and **v_balance** the signed Orchard value balance. A positive v_balance releases net value from Orchard; a negative value moves net value into Orchard. For every non-coinbase transaction:

**fee = T_in + v_balance - T_out >= 0**

The private proof and binding signature enforce the relationship between hidden note values and v_balance. Nodes also check transparent values, signed arithmetic and fee ranges. For a transaction without Orchard, v_balance is zero and the equation reduces to ordinary transparent accounting.

## 7.2 Public Pool Accounting

Although individual private note values are concealed, the total value entering and leaving the pool is public. Let **P** be the aggregate pool balance before a transaction:

**P_after = P_before - v_balance**

Both pool balances must remain between zero and the maximum money amount. Nodes reject a transition that underflows, overflows or leaves that range. The pool begins empty and is updated in transaction order. It cannot create additional currency by moving value between its transparent and private forms.

## 7.3 Worked Examples

The following examples use NITO rather than nitoshi. All amounts are exactly representable in the smallest unit.

| Payment | Public accounting |
|---|---|
| Shield 99.99 NITO from a 100 NITO transparent input | T_in = 100, T_out = 0, v_balance = -99.99. Fee = 0.01; pool increases by 99.99. |
| Pay privately, funding a 0.01 NITO fee from private notes | T_in = 0, T_out = 0, v_balance = 0.01. Fee = 0.01; pool decreases by 0.01. Recipient and change values remain concealed. |
| Unshield 20 NITO with a 0.01 NITO fee paid privately | T_in = 0, T_out = 20, v_balance = 20.01. Fee = 0.01; pool decreases by 20.01. |

Transactions may also combine transparent change with private outputs, or spend several notes in one bundle. The same equation applies to the totals, regardless of how the wallet selects individual inputs and change.

## 7.4 Why Hidden Values Still Balance

For action **i**, let **d_i** be value spent minus value created and **r_i** its blinding scalar. Its value commitment has the form **C_i = d_i * G + r_i * H**, where G and H are fixed Pallas group generators with no known discrete-log relation. If v_balance equals the sum of the d_i values:

**sum(C_i) - v_balance * G = sum(r_i) * H**

The binding signature authorizes the transaction under the key derived from the left-hand side. Circuit constraints tie each commitment to valid, range-checked note values. Together they enforce balance without publishing those values; a public pool total alone cannot do so. [3]

# 8. Transaction Commitments and Encoding

## 8.1 Effects and Authorization

Nito separates private **effects**, which describe the requested ledger change, from private **authorization**, which proves and signs that change. Effects include the bundle version, flags, anchor, signed value balance and action data. Authorization includes one spend signature per action, the Halo 2 proof and a binding signature.

The extended transaction serialization uses flags for transparent witness, private effects and private authorization. Private effects follow transparent inputs and outputs; transparent witness and private authorization follow when present; lock time remains at the end. Unknown flags and private authorization without effects are rejected.

This is a Nito transaction format. Sharing Orchard cryptography with another network does not make transaction encodings, identifiers or signatures interchangeable between those networks.

## 8.2 Transaction and Witness Identifiers

The transaction identifier, **txid**, hashes the stripped serialization. It includes Orchard effects but excludes transparent witness and private authorization. The witness identifier, **wtxid**, hashes the complete serialization:

> **txid = SHA256d(stripped transaction)**
>
> **wtxid = SHA256d(full transaction)**

The header's transaction Merkle root commits to txids. The SegWit witness commitment in the coinbase commits to the witness Merkle root and reserved value, thereby covering proof and signature bytes as well. Effects cannot be detached from the transaction, and authorization cannot be discarded without affecting the full commitment and validity.

## 8.3 Binding Signatures to the Payment

Transparent signature hashes for a transaction containing Orchard include a commitment to its private effects, including for legacy, SegWit and Taproot signing paths. This prevents private recipients, commitments or value balance from being changed independently of the transparent authorization.

Orchard signatures use a Nito-domain-separated hash of the stripped transaction with transparent input scripts cleared. This commits to the transaction's spending references, outputs and private effects while allowing transparent signatures to be added without changing the Orchard signing message. Transparent-only transactions retain their established signature-hash behavior.

## 8.4 Encoded Sizes

Each action carries a 32-byte ephemeral key, a 580-byte recipient ciphertext and an 80-byte outgoing ciphertext. RedPallas spend and binding signatures are each 64 bytes. For **n** actions, the required proof length is:

> **proof_bytes(n) = 2,720 + 2,272 * n**

A two-action proof is 7,264 bytes; a 50-action proof is 116,320 bytes. These figures exclude action effects, signatures, length prefixes and the rest of the transaction. A correct length is only a structural requirement: all encoded points, proofs and signatures still require validation.

# 9. Validation and State Transitions

## 9.1 Structural and Contextual Checks

A node first checks the transaction structure, supported versions and flags, amount ranges, action count, ciphertext lengths and authorization sizes. Duplicate transparent inputs and duplicate nullifiers within a bundle are invalid. A private transaction requires its complete authorization; a coinbase may not contain private data.

Context then matters. Transparent inputs must exist and satisfy maturity and lock conditions. An Orchard anchor must be recognized by the current chainstate, and each nullifier must be unspent there. The fee must be nonnegative and within range. A transaction that was valid on one branch may cease to be valid after a reorganization.

Nodes verify transparent scripts, RedPallas spend signatures, the binding signature and the Halo 2 proof. These checks enforce complementary requirements. Passing a size check or using a recognized anchor cannot substitute for cryptographic verification.

## 9.2 Connecting and Disconnecting Blocks

For a confirmed Orchard transaction, nodes append the new note commitments, record the nullifiers, add the resulting tree root as an accepted anchor and update the pool balance. Transparent UTXOs are updated in the same transaction order. The aggregate block must satisfy its proof of work, commitments, weight, signature-operation limit and reward constraint.

Undo information records the previous private tree frontier, root, commitment count and pool balance, together with the added nullifiers and commitments. Disconnecting a block removes the corresponding effects and restores the earlier state. Roots and nullifiers from a disconnected branch must not silently remain valid on the replacement branch.

The commitment tree is append-only along one active history, but a reorganization changes that history. A wallet must keep its note positions and Merkle paths consistent with the chain it is actually using.

## 9.3 Resource Protection

Private proof verification costs more computation than parsing a transaction. Nito Core applies inexpensive checks before expensive proof verification, includes the verification floor in policy virtual size and uses a bounded cache of successful proof results. Invalid proofs and verifier failures are not cached as successful validation.

Unsolicited private transactions received from peers share a verification budget. In the referenced implementation, it replenishes at 55,000 policy units per second and allows a burst of 110,000 units. This is a peer-to-peer admission policy; block validation and locally submitted transactions are not governed by that rate limiter.

The maximum block weight and action count bound accepted data and associated work. Policy limits can reduce abuse, but they do not establish a fixed transactions-per-second rate. Transaction mix, proof construction, hardware, storage and network conditions all affect practical throughput.

## 9.4 Indexes Are Not Consensus Authority

Wallet indexes accelerate discovery and recovery. They do not authorize spending or override chain validation. A missing or lagging index cannot make an invalid note spend valid. Consensus state and wallet discovery data have distinct responsibilities, even when both are maintained by the same node.

# 10. Privacy and Public Information

## 10.1 What Each Payment Reveals

| Payment path | Public information |
|---|---|
| Transparent to transparent | Spending references, output values, spending conditions and their public links |
| Transparent to Orchard | Transparent inputs and any change; net value entering the private pool |
| Orchard to transparent | Transparent destinations and amounts; net value leaving the pool |
| Orchard to Orchard | Bundle presence, action count, commitments, nullifiers, anchor, encrypted payloads, proof, signatures and fee |

In all cases, the transaction identifier and its inclusion in a block are public. Private recipient addresses and individual note values are not published in plaintext. The proof hides which eligible earlier note commitment is being spent, subject to the security of the construction.

## 10.2 Correlation at Transparent Boundaries

An observer may compare amounts and timing when value enters and leaves Orchard. Shielding an unusual amount and shortly afterward unshielding a similar amount gives that observer evidence even though the intervening note is encrypted. Repeated transparent address use also exposes relationships outside the private pool.

The set of plausible private activity depends on actual participation, usage patterns and external information. A large tree of commitments does not mean every note is equally plausible for every payment. Privacy should not be reduced to a numerical anonymity guarantee inferred from the action count or tree size alone.

## 10.3 Network and Counterparty Information

Ledger privacy does not by itself hide the network origin of a broadcast. Peers or observers may correlate connection information and timing. Tor and supported proxy transports can reduce exposure of a device's direct network address, but do not eliminate every traffic-analysis risk.

Recipients know the payments they receive. Merchants, exchanges and other counterparties may keep their own records. A private on-chain transfer does not erase those records or make a publicly shared invoice confidential.

## 10.4 Viewing Authority and Device Security

A viewing key can reveal private activity within its scope without granting spending permission. Sharing one is therefore a disclosure decision. An explorer using public chain data alone cannot display concealed values or identify private recipients; a receiving service needs its own authorized wallet view to credit deposits.

Malware can read information while a wallet is open, alter a destination before signing or steal unprotected keys. Wallet-file encryption protects stored secrets under its security assumptions; it cannot protect information already disclosed to another party or recover funds after spending authority has been stolen.

Private payments improve confidentiality on the ledger. Their practical protection still depends on key handling, device integrity, the payment path and the information the parties disclose elsewhere.

# 11. Wallet Keys and Addresses

## 11.1 Spending and Viewing Authority

Spending keys authorize transfers. Orchard full viewing keys allow discovery of incoming notes and relevant outgoing activity, including change, without authorizing a spend. Nito Core maintains external and internal Orchard branches so the wallet can track receiving activity and its own change.

The receiver encoded in a Bech32x address is public information suitable for sharing with a sender. It is not a full viewing key. Nito encodes a version and receiver type followed by a 43-byte Orchard receiver, converts the payload to five-bit symbols and protects it with a Bech32m checksum. The distinguishing tag produces the mainnet prefix **nito1x**. This is a direct private receiver format, not a transparent witness program.

## 11.2 Deterministic Recovery Material

A native v4 wallet can derive transparent and Orchard authority from a common recovery root. A BIP39 English recovery phrase records the seed material in a human-readable form. Recovery coverage still depends on how the wallet was created and whether keys were imported separately; Section 12 explains these distinctions.

BIP39 seed derivation uses PBKDF2-HMAC-SHA512 with 2,048 iterations and a 64-byte result. Its inputs are the normalized mnemonic and a salt consisting of the string **mnemonic** followed by the normalized optional passphrase. Nito applies Unicode NFKD normalization. [5]

An optional **BIP39 passphrase** changes the wallet derived from a phrase. Omitting it means an empty passphrase. It is a separate secret, not another recovery word. A different valid passphrase, including a typing error, derives a different valid wallet rather than reporting that the intended wallet password was wrong.

## 11.3 Two Different Password Roles

The BIP39 passphrase is distinct from the password that encrypts the wallet file. Nito Core does not store the BIP39 passphrase and cannot recover or reset it. Phrase-based recovery requires the original phrase and the same passphrase, if one was used.

A wallet-file backup preserves the derived seed record without storing that passphrase. Restoring the file can therefore restore its saved authority without re-entering the BIP39 passphrase; the file's own encryption still applies. A BIP39 passphrase is not an extra lock on an already-derived seed stored in a wallet file.

## 11.4 Viewing an Encrypted Wallet

The Qt wallet asks for authentication before displaying an encrypted wallet's information. It can then retain a viewing session while relocking spending authority. Operations that require private keys ask for authorization again and relock when complete. Explicitly locking or closing the wallet ends the viewing session.

This distinction makes balances and history usable without leaving spending authority continuously unlocked. Viewing an open wallet nevertheless exposes its displayed information to anyone with access to that session or device.

# 12. Backups and Recovery

## 12.1 What a Backup Covers

| Recovery source | Scope |
|---|---|
| Complete wallet-file backup | Keys and descriptors present at backup time, imported keys, stored Orchard authority, labels, address book, encryption state and local metadata |
| Native v4 recovery phrase, with BIP39 passphrase if used | Deterministic transparent and Orchard authority derived from that root; recoverable funds and history from chain data |
| Phrase created when adding Orchard to an existing v3 wallet, with BIP39 passphrase if used | Added Orchard authority only; earlier transparent and imported keys remain dependent on the original wallet backup |

The native phrase scope is called **full_wallet**. The phrase supplied when Orchard is added to an older wallet has scope **orchard_only**. These names describe recovery coverage, not two different assets or blockchains.

## 12.2 Existing Wallets

Existing transparent keys are not reconstructed from a newly added Orchard phrase. The explicit Orchard upgrade creates new private authority and a pre-upgrade backup; it does not silently convert arbitrary older keys into a new deterministic root.

Keep the original v3 wallet backup for its transparent and imported keys, and create a current complete backup after adding Orchard. A pre-upgrade file does not contain the Orchard keys created later. A new recovery phrase does not replace the older file's authority.

Imported keys require particular care even in a native wallet. A deterministic phrase covers keys derived from its root, not every unrelated key later imported into the same wallet. Labels, comments and address-book entries are local information and cannot be reconstructed from blockchain consensus data alone.

## 12.3 Recovering History

Possessing the correct key material and reconstructing the wallet's history are separate tasks. Recovery scans chain data for the relevant transparent outputs and encrypted Orchard notes, identifies spends and reconstructs the state required to use them. Completion requires the appropriate indexes to reach the requested chain target.

Transparent address discovery is bounded. If an application deliberately used addresses beyond the normal derivation search range with a large unused gap, recovery may require a larger search range. A phrase does not encode an unlimited list of previously used address indexes.

## 12.4 Keeping Recovery Material

Store recovery phrases and any BIP39 passphrase offline, with protection against loss and unauthorized access. Keep wallet-file backups as well, especially where imported keys or an older wallet are involved. Anyone with sufficient recovery material can exercise the authority it covers.

Use the wallet's backup function to obtain a consistent file. A backup can be checked by restoring it in a separate local wallet or isolated data directory while retaining the original. Recovery does not require sending a phrase, spending key or wallet file to a public index server or support operator.

# 13. Indexes, Storage and Pruning

## 13.1 Chainstate and Historical Data

Current chainstate answers whether a proposed spend is valid now. Historical blocks allow a wallet to discover past activity. Indexes organize that history for efficient lookup. These are different storage needs: a node can have a valid current chain while a newly enabled wallet index is still catching up.

An archival node retains the historical blocks needed for reconstruction. A **pruned node** validates blocks and later deletes older block files to limit their storage. Pruning does not weaken the rules used to validate new blocks, but it limits which past data can be read from that node.

## 13.2 Orchard Index

The Orchard index stores public compact action data, commitment and anchor information, and complete public transactions containing Orchard. This supports note discovery, private transaction history and Merkle-path reconstruction even after the corresponding old block files are pruned.

It does not contain wallet spending keys, full viewing keys or decrypted private notes. Wallets perform the private recognition and decryption locally. Index data is additional storage beyond the configured block-pruning target.

Orchard indexing is enabled by default, including on newly configured pruned nodes. The index must have processed the relevant history before blocks are deleted. With that retained index and a synchronized wallet, private address generation, balance display and spending can continue after older block files have been removed.

An explicitly disabled or incomplete Orchard index makes the corresponding private wallet operations unavailable. The node does not substitute an obsolete note path or present an incomplete recovery as successful.

## 13.3 Complete Phrase Recovery

Automatic complete recovery from a phrase requires an unpruned node and a synchronized transparent-history index. If Orchard authority is involved and the recovery target reaches private activation, the Orchard index is required as well. Recovery must cover the requested chain target and account for reorganizations.

The retained Orchard index can preserve private discovery on a pruned node, but it does not reconstruct missing transparent history. Importing a phrase against incomplete pruned history is therefore refused for complete recovery rather than silently returning a partial wallet.

## 13.4 Missing History

Changing a storage setting does not make deleted blocks reappear. If required history is absent, recovery needs a suitable archival data source and an explicit reconstruction or resynchronization operation. A recovery error does not automatically authorize a full redownload.

Ordinary spending by an already synchronized wallet should be distinguished from restoring a wallet on a new machine. The first may work with retained state and indexes; the second may require history no longer present locally. Maintaining both current wallet backups and recoverable chain access avoids confusing these two situations.

# 14. Activation and Network Operation

## 14.1 Mainnet Boundary

Orchard validation begins at **block 1,165,432**. Let H denote that height. Blocks below H cannot contain an Orchard bundle. From H onward, both transparent-only blocks and blocks containing valid Orchard transactions are allowed. Activation enables private payments; it does not require a private transaction in every block.

The mempool evaluates transaction eligibility for the next block. At tip H - 1, it can therefore accept a valid Orchard transaction for candidate block H. Wallet availability also depends on the active chain, wallet authority and index readiness. These checks prevent private operations from being presented as ready merely because compatible software has been installed.

## 14.2 Compatible Peers

A peer is considered capable of the private protocol when it advertises protocol version **70017 or later** and the **NITO_PRIVATE** service capability. After activation is observed, peers that do not satisfy both requirements are disconnected. A future version number alone is not sufficient without the required capability.

A transparent-only activation block can be received from an older peer while the local tip is still below H. A block carrying Orchard requires a capable source. Once activation has been observed, the node persists that network boundary: a restart or a reorganization below H does not reopen connectivity to incompatible peers.

Transaction validity still follows block height. Persisting the peer boundary does not permit Orchard in a block below H. If compatible peers are unavailable, the node can remain isolated rather than use a peer unable to validate the private protocol.

## 14.3 Participants and Services

Users choose whether to use private payments, but all validating nodes need software capable of applying the activated rules. Pools, exchanges, explorers and wallet services must account for complete transaction serialization and private authorization even if their own deposits or payouts are transparent.

The mainnet peer-to-peer port is **8820**. The default mainnet RPC port is **8825**; RPC is a local management interface, not a public payment broadcast service to expose without authentication and access controls. Peers discover and relay transactions and blocks; each full node retains responsibility for validation.

Mainnet, testnet, Signet and regtest are separate environments. Their histories, coins and address-network parameters are not interchangeable. Test settings do not accelerate or redefine mainnet activation.

## 14.4 Continuity

The upgrade extends the existing Nito chain and retains existing transparent balances. It does not require exchanging coins for another token. Compatible software and network adoption are necessary for following the activated chain.

Official release information and supported wallet procedures are published through the Nito Core repository.

# 15. Security and Protocol Evolution

## 15.1 Proof-of-Work Assumptions

The ordering of transactions depends on competition among miners and the work accumulated by valid branches. An attacker with sufficient hash power can censor transactions, reorganize recent history or attempt to reverse its own payments. Honest nodes still reject invalid signatures, excessive rewards and invalid private proofs regardless of the work attached to them.

SHA-256 hardware availability does not establish how much hash power secures Nito or how widely it is distributed. A shorter target interval changes the pace of expected blocks; it does not by itself provide the same security as a particular number of confirmations on another network. Payment acceptance should consider accumulated work, observed network conditions and the consequences of reversal.

## 15.2 Cryptographic and Implementation Assumptions

Transparent authorization relies on the relevant signature and hash constructions. Orchard additionally relies on note commitments, nullifiers, encryption, proof soundness and binding authorization. A circuit flaw or a failure to bind all transaction effects can threaten value conservation even if the visible arithmetic appears correct.

Hidden note values limit public inspection of individual holdings. The public pool balance is useful for aggregate accounting, but does not independently audit every private note. Independent implementation review and cryptographic analysis remain necessary. No statement of protocol parameters is a proof that an implementation contains no defects.

Compromised software, poor randomness or exposed recovery material can defeat otherwise correct cryptography. The design does not claim resistance to a cryptographically relevant quantum computer capable of breaking the elliptic-curve assumptions used for signatures and private payments.

## 15.3 Availability and Synchronization

Peer isolation, denial of service, inaccurate clocks, storage exhaustion and unavailable recovery history can prevent useful operation without creating a valid attack on the underlying mathematics. Resource limits and peer policies reduce some exposures but cannot guarantee uninterrupted availability.

Nito Core also uses synchronization safeguards, including checkpoints, minimum-chainwork expectations and an assumed-valid historical block for eligible script-check optimization. These are implementation assumptions relevant to how a node initially synchronizes, not substitutes for valid proof of work. Archival validation and review of node settings remain distinct from trusting the balance displayed by a remote service.

## 15.4 Changing the Rules

Consensus rules are enforced by the software participants choose to run. A change to issuance, accepted transaction semantics or the Orchard circuit requires defined rules and coordinated adoption. Nodes following incompatible rules can disagree about the valid chain.

There is no on-chain vote in this upgrade that automatically rewrites the protocol. Future changes need an explicit technical specification, review of their security implications and a clear transition for existing users. The objective is verifiable payment ownership and conservation of value; neither the protocol nor this paper guarantees future market value or economic adoption.

# 16. Glossary

| Term | Meaning |
|---|---|
| Action | Orchard structure combining a spend side and an output side, potentially using dummy notes. |
| Anchor | An accepted root of the note commitment tree used by a private proof. |
| Bech32x | Nito address encoding for a direct Orchard receiver, using a Bech32m checksum. |
| Block height | Position in the chain, starting at zero for genesis. |
| Coinbase | First transaction in a block; claims up to the subsidy and transaction fees. |
| Commitment | A cryptographic binding to information that can conceal its contents. |
| Confirmation | Inclusion in a block; additional confirmations add blocks above that inclusion. |
| Consensus | Rules determining which transactions, blocks and state transitions are valid. |
| Difficulty | A measure that varies inversely with the proof-of-work target. |
| Full node | Software applying the network's validation rules and maintaining chainstate. |
| Halo 2 | Zero-knowledge proving system used to verify Orchard circuit constraints. |
| Mempool | A node's local collection of transactions awaiting confirmation. |
| Nitoshi | One hundred-millionth of one NITO. |
| Note | Private value assigned to an Orchard recipient with cryptographic randomness. |
| Nullifier | Public spend identifier preventing a private note from being spent twice. |
| Orchard | The private payment construction integrated into the Nito ledger. |
| Policy | Local rules for relaying, storing or selecting transactions, distinct from consensus validity. |
| Pruning | Deletion of older block files after validation while retaining current chainstate. |
| Reorganization | Replacement of part of the active chain with another accepted branch. |
| Shielding | Moving transparent value into private Orchard notes. |
| Spending key | Secret authority used to authorize payments. |
| UTXO | An unspent transparent transaction output. |
| Value balance | Signed public net value released from or deposited into Orchard. |
| Viewing key | Authority to inspect private activity within its scope without permission to spend. |
| Witness | Depending on context, transaction authorization data or secret data used in a proof. |
| Zero-knowledge proof | Evidence that specified constraints hold without disclosing the private witness. |

# 17. References

## Nito Protocol

**1. Nito Network. Nito Core.** Implementation reference: commit **69b51fff6d9f4827cfbc3b74e0604541ba47cf8e** on the public v4.0 source branch. The links below identify this revision so that its parameters and behavior remain inspectable as the project evolves.

Protocol source. Mainnet parameters: chain parameters. Work and target adjustment: proof of work. Subsidy, block connection and validation: validation.

Private accounting and state: Orchard state transitions. Encoding and proof lengths: private transaction structures. Accepted circuit and cryptographic verification: Orchard verifier. Fee sizing: transaction policy.

User documentation: wallet backups and recovery, private addresses and wallet behavior, security and privacy model.

## Cryptographic Foundations

**2. Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System.** Proof-of-work ordering and the double-spending problem. Original paper.

**3. Zcash Protocol Specification and ZIP 224.** Orchard notes, commitments, actions, nullifiers, encryption and authorization. Protocol specification; Orchard shielded protocol.

**4. The Halo 2 Book.** Proof-system design, circuit constraints and polynomial commitments. Technical documentation; proving system.

**5. Bitcoin Improvement Proposal 39.** Mnemonic encoding and optional passphrase-based seed derivation. BIP 39.

**6. Bitcoin Improvement Proposals 141 and 143.** Segregated Witness, weight, witness commitments and signature hashing. BIP 141; BIP 143.

**7. Bitcoin Improvement Proposals 340, 341 and 342.** Schnorr signatures, Taproot and Tapscript. BIP 340; BIP 341; BIP 342.

**8. Bitcoin Improvement Proposals 173 and 350.** Bech32 and Bech32m address encoding. BIP 173; BIP 350.

Bitcoin and Zcash sources explain shared constructions. Their network parameters, transaction formats and later upgrades do not define Nito consensus. Nito-specific integration and recovery behavior are documented in Reference 1.

nito.network | Nito Core