TensorCash Settlement Contracts Whitepaper: Native Contracts and the Tapscript Extension

Imosuke Takakuni[1]

2 May 2026

ABSTRACT

TensorCash is a Bitcoin-fork chain that ties block production to verified language-model inference and settles issued assets on the same UTXO ledger as the native unit. This document covers the native settlement contracts implemented in the core node: spot atomic swaps, collateralised repurchase agreements, and two-sided initial-margin forward / option-style delivery-versus-payment contracts. Asset issuance is a separate protocol layer; it is not part of the native settlement contract surface described here.

The core design choice is deliberately conservative. TensorCash does not add a general-purpose virtual machine, contract storage, loops, oracle callbacks, or arbitrary execution. Settlement enforcement remains Bitcoin-style Script. The financial covenant primitive underpinning the contracts is `OP_OUTPUTMATCH`: a Tapscript-only check that asks whether the spending transaction contains an output with a specified spendable script hash, amount, and, for asset outputs, asset identifier. It is intentionally cheap and robust: it performs bounded output matching over the current transaction, does not execute recipient scripts, does not consult transaction history, and does not maintain mutable state.

Taproot gives these contracts confidentiality before enforcement. A funded vault appears on chain as a Taproot output; its leaves and exact contract terms are hash commitments until a script path is spent. The public UTXO set does not reveal a full forward or repo term sheet by itself. The parties' wallets know the terms from the negotiated offer, acceptance, PSBT metadata, and local contract registry; observers learn only the branch and constraints that are revealed at spend time.

1. "Imosuke Takakuni" is a pseudonym. Contact: takakuni@tensorcash.org

TABLE OF CONTENTS

I. What "Contract" Means in TensorCash

In the legal sense, a contract is a promise enforced by law. On TensorCash, a native settlement contract is a promise enforced by consensus: a set of UTXOs and spend paths that restrict which later transactions may consume those UTXOs. The restriction is not "run arbitrary program X". It is narrower: spend this vault only if the transaction also pays the agreed output, waits until the agreed height, or carries the agreed signatures.

That distinction matters. A TensorCash contract is not a public storage object whose fields can be read from chain state. Until a tapleaf is used, the public chain normally sees only a Taproot output key. The contract terms are known to the parties from the off-chain offer and from wallet registry metadata, and they are committed into the taproot tree. When a branch is exercised, the revealed script and the spending transaction prove exactly which condition was enforced.

The settlement machinery shipped in the core node has three native shapes. A spot swap is a single atomic PSBT that both wallets sign only after the complete input and output set is present. A repo is a collateral vault with a repayment-release leaf and a maturity-default leaf. A forward is an initial-margin DvP state machine built from two margin vaults and two conditional escrows. All three settle on the regular UTXO set, and all asset outputs remain subject to the asset protocol's TLV and validation rules.

II. Spot Atomic Swap

The simplest native contract is a spot atomic swap: one transaction in which two parties exchange assets with no half-settlement state. Each party contributes the inputs it controls and the output paying the asset it is delivering. Each party signs only after the merged PSBT contains the agreed destination outputs. If either leg is missing, the transaction is not fully signed; if both legs are present and signed, settlement occurs in a single block.

Spot swaps do not need a taproot vault. Their atomicity comes from ordinary transaction validity and signature consent over the final transaction. For issued assets, the relevant outputs carry asset TLVs, and regulated assets must satisfy the asset protocol's transfer requirements in the same transaction. The settlement contract layer does not create assets and does not loosen asset policy; it only coordinates the exchange of already-valid asset outputs.

III. The Repurchase Agreement

A repo is a collateralised loan. The borrower posts collateral and receives principal; the lender receives principal plus interest if the borrower repays, or takes the collateral after maturity if the borrower does not. TensorCash implements this as a collateral vault rather than as an account relationship.

The opening PSBT funds the borrower's principal payout and the collateral vault atomically. There is no separate lender cash vault in the repo tree. The collateral vault has two relevant paths.

The first path is the repayment release. It requires the borrower's signature and one or more `OP_OUTPUT_MATCH` checks proving that the spending transaction contains the negotiated repayment output or outputs to the lender's repayment destination. If principal and interest are the same asset, the wallet can merge them into one output-match check. If they differ, it emits separate checks. The covenant pins the lender's repayment outputs while still allowing fee inputs, change, and unrelated outputs.

The second path is the lender's default claim. After the maturity height, if the borrower has not repaid through the repayment leaf, the lender can sweep the collateral vault through a `CHECKLOCKTIMEVERIFY`-guarded signature path. The chain does not price the collateral and does not compute interest dynamically; both are negotiated off-chain and encoded into the opening and repayment terms.

IV. Forward and Option-Style DvP Contracts

The implemented forward contract is a two-sided, initial-margin-capped DvP primitive. It is general enough to express plain forwards, covered option-like payoffs, and bespoke bilateral delivery trades by changing the two deliverable legs, the two margin legs, the deadlines, and the optional upfront premium `P0`.

The term sheet contains four asset legs. The long party has a `deliver_leg`, a `margin_leg`, a margin return destination, and a settlement receive destination. The short party has the same fields. The term sheet also contains `deadline_short` (`T`), `deadline_long` (`T+K`), optional `safety_k` and `reorg_conf` wallet parameters, and an optional `premium_upfront` paid in the opening transaction. `P0` is an upfront economic term. It is not a later mark-to-market cash-settlement branch.

i. Opening the Contract

`forward.build_open` creates both initial-margin vault outputs atomically. The long wallet funds the long margin vault and, when applicable, the long-paid premium output; the short wallet augments the PSBT with the short margin funding. Each margin vault can hold the native unit or an issued asset. For non-native margin or delivery legs, the output carries an asset tag and the asset protocol validates it like any other asset output.

The opening transaction commits to the taproot trees for both margin vaults. It also fixes the escrow scripts that may be created later by a self-delivery spend. Those escrow outputs are not pre-funded at open. They are created only when one party spends its own margin vault through a self-delivery leaf and places its deliverable into escrow.

ii. Cooperative Close

The cooperative close spends both margin vaults through their cooperative leaves. It requires both parties' signatures and four output-match checks:

- the short party's deliverable paid to the long party's settlement destination
- the long party's deliverable paid to the short party's settlement destination
- the long party's margin returned to the long party's margin destination
- the short party's margin returned to the short party's margin destination

This is the fully cooperative DvP close. It is not a two-output cash-settlement branch. The current native covenant tree does not contain a distinct cash-settlement tapleaf that only returns adjusted margins. Any cooperative close must satisfy the covenant leaves actually committed at open, or the parties must roll into a new contract with new terms.

iii. Unilateral DvP Through Escrow

The important forward mechanism is the unilateral path. It avoids a fragile physical-delivery rendezvous by making the first mover deliver into an escrow and making the second mover claim that escrow only by delivering the opposite leg in the same transaction.

If the short party acts first, it spends the short margin vault through the `B-SELF` path. That spend must return the short margin to the short margin destination and must create `B_ESCROW` containing the short party's deliverable. The short party signs this path. The long party can then claim `B_ESCROW` only through a script that requires an `OP_OUTPUTMATCH` proof that the same spending transaction pays the long party's deliverable to the short party's settlement destination. In the same transaction, the long party may also spend its own margin vault through the counter-delivery path to recover its margin.

If the long party acts first, the mirror path applies. After `T`, the long party can spend the long margin vault through `A-SELF`, return its own margin, and create `A_ESCROW` containing the long party's deliverable. The short party can claim `A_ESCROW` only by paying the short party's deliverable to the long party in the same transaction.

The escrows have refund leaves. `B_ESCROW` refunds to the short party after `T+K`; `A_ESCROW` refunds to the long party after `T+K`. This means a party that puts its asset into escrow is not forced to leave it there forever if the other side refuses to complete the DvP.

iv. Deadlines, Margin, and Limited Recourse

The deadlines create symmetric incentives. The short party is expected to act by `T`. If it does not, the long party can sweep the short party's initial margin at `T`. The long party can also self-deliver into `A_ESCROW`; if the short party still refuses to deliver the opposite leg, the long party refunds its own escrowed asset after `T+K`.

Conversely, if the short party self-delivers into `B_ESCROW` and the long party refuses to claim by delivering the opposite leg, the short party refunds `B_ESCROW` after `T+K` and can sweep the long party's initial margin. The losing party's exposure is limited to the margin it posted and any asset it deliberately placed into a refundable escrow. There is no variation margin, daily mark-to-market, oracle price, or external enforcement dependency.

That absence of variation margin is deliberate. The contract is limited recourse so it can be enforced entirely by Script, output matching, signatures, and timelocks. The chain never decides what the forward is "worth" at maturity. The parties decide the margin, deliverables, premium, and deadlines before funding; after funding, consensus enforces only those committed terms.

v. Confidentiality

Before a forward branch is spent, the public chain sees Taproot outputs, not a readable contract. The deliverables, margins, deadlines, and leaf structure are committed in the taproot tree and stored by the parties' wallets. A spend reveals the branch needed for enforcement, and the transaction outputs reveal the concrete settlement required by that branch.

V. The Tapscript Extension: Implemented Opcodes

The settlement contracts use a deliberately small script extension. The financial covenant primitive is `OP_OUTPUTMATCH`, exposed in native and asset-aware forms. The codebase also implements ML-DSA signature opcodes for post-quantum signing paths, but those are key-authentication primitives rather than the covenant mechanism that makes repo and forward settlement work.

i. OP_OUTPUTMATCH_NATIVE

`OP_OUTPUTMATCH_NATIVE` is the native-unit covenant opcode. In Tapscript it pops a 32-byte tap-match script hash and an 8-byte little-endian amount. It pushes true only if the spending transaction contains a spendable output whose script hash matches and whose native amount equals the requested amount, with no asset TLV attached.

The opcode is intentionally narrow. It is Tapscript-only. It rejects malformed operand sizes, zero amounts, and native amounts outside `MAX_MONEY`. It scans the current transaction outputs through the signature-checker interface and short-circuits once a match is found. It does not pin output order, does not constrain inputs, and does not prevent ordinary fee and change handling.

ii. OP_OUTPUTMATCH_ASSET

`OP_OUTPUTMATCH_ASSET` is the asset-aware form. It pops a 32-byte tap-match script hash, a 32-byte asset identifier, and an 8-byte little-endian asset amount. It pushes true only if the spending transaction contains a spendable output with a matching script hash and an `AssetTag` TLV carrying the same asset id and amount.

This is what lets a repo repayment leaf require repayment in an issued asset, and what lets a forward leaf require a specific asset delivery or asset margin return. Invalid asset identifiers are not a settlement mode; asset outputs still have to satisfy the asset protocol's normal validation rules.

iii. OP_CHECKMLSIG and OP_CHECKMLSIGVERIFY

`OP_CHECKMLSIG` and `OP_CHECKMLSIGVERIFY` verify ML-DSA signatures in Tapscript. They are implemented for post-quantum signature paths and PSBT signing support. They do not add contract storage, covenants, or a VM. Long-dated vaults may include ML-DSA signing paths when wallets want post-quantum key protection, but the repo and forward covenant logic described above is enforced by `OP_OUTPUTMATCH`, signatures, and timelocks.

iv. Witness Version 2

Witness-version-2 Taproot is used for standard relay of larger post-quantum witness elements. It raises the standard stack-element ceiling for the larger ML-DSA public keys and signatures while keeping the new behavior opt-in and script-path oriented. Contracts that do not need ML-DSA can continue to use ordinary Taproot witness-v1 outputs.

v. What Was Not Added

TensorCash does not add arbitrary computation, contract storage, loops, off-chain oracle queries, or a script-level zero-knowledge verifier to the implemented opcode table described here. Asset compliance remains part of the asset transaction-validation layer. Settlement contracts rely on output matching because it is small enough to review and cheap enough to bound with simple policy limits, including the per-input `OP_OUTPUTMATCH` cap.

VI. Wallet RPC and Contract Book Management

The wallet exposes contract-aware RPCs for proposing, importing, accepting, building, signing, and closing native contracts. Repo, spot, and forward each have lifecycle RPCs for their own ceremonies, while `contract.list` and `contract.status` provide a unified view of stored wallet contract records.

For taproot vaults, the wallet stores metadata that ordinary chain state does not reveal: contract id, role, taproot spend data, leaf descriptors, signing keys, and relevant vault outpoints. Later lifecycle RPCs use that registry to choose the correct leaf and attach the correct PSBT signing intent. This is why the wallet can show a contract book without reconstructing every term from public transaction history.

The forward RPC surface follows the implemented state machine. `forward.build_open` creates the two margin vaults. `forward.build_coop_close` constructs a cooperative four-output DvP close. `forward.build_self_delivery` spends the local margin vault to create `A_ESCROW` or `B_ESCROW`. `forward.build_escrow_claim` claims an escrow while delivering the opposite leg. `forward.build_escrow_refund` refunds an unclaimed escrow after `T+K`. `forward.build_im_timeout` sweeps the counterparty's margin when the relevant deadline has passed.

VII. Composition: How the Pieces Fit

The native settlement layer composes three simple ideas. Spot swaps are one-shot atomic transactions. Repos use one collateral vault with repayment and default paths. Forward / option-style contracts use two initial-margin vaults plus escrow leaves so either side can force fair DvP or collect the posted margin when the counterparty refuses to act.

All issued-asset legs are ordinary asset outputs subject to the asset protocol. The settlement contract layer does not create assets, redefine issuer policy, or bypass transfer validation. Its job is narrower: make sure the transaction that spends a vault also contains the exact output required by the negotiated contract.

VIII. Security Posture

Half-settled trades. Spot swaps require both sides' signed inputs in one transaction. Repo and forward vaults use output-match checks so a vault spend cannot omit the repayment, delivery, escrow, or margin output required by that branch.

Delivery-rendezvous failure. Forward unilateral delivery routes the first deliverable into an escrow. The counterparty can take that escrow only by delivering the opposite leg in the same transaction. If it does not, the first mover gets a refund after the long deadline.

Limited recourse. Forward contracts do not depend on variation margin, price oracles, or off-chain enforcement. The enforceable loss is bounded by the posted initial margin and the assets voluntarily placed into refundable escrow.

Hidden terms before enforcement. Taproot keeps the contract leaves hidden until a branch is spent. Public observers cannot read the full contract from the UTXO set alone.

Covenant griefing. `OP_OUTPUTMATCH` matches only spendable output script families and exact amounts. Policy caps the number of output-match opcodes per input and the maximum covenant transaction output count, keeping validation cost bounded.

Post-quantum signing risk. ML-DSA opcodes provide an implemented path for wallets that want post-quantum signature checks on long-dated vaults. They are separate from the output-match covenant primitive.

This whitepaper accompanies the TensorCash Asset Protocol whitepaper for asset TLV and transfer-validation details. The implemented settlement contract code lives primarily in `services/core-node/bcore/src/wallet/rpc/contracts.cpp`*, with the output-match interpreter logic in* `services/core-node/bcore/src/script/interpreter.cpp` *and policy bounds in* `services/core-node/bcore/src/policy/`*.*