

Tuliproot: Technical Specification Draft

Michael Tidwell
michael@tidwell.io
tuliproot.com

Draft 0.6 | October 23, 2026

Status and terminology

This companion to the short whitepaper retains the detailed design developed through draft 0.5. It is a specification draft: conceptual records and verifier checks are proposed, while wire bytes, policy templates, and failure rules remain unfinished. It is not an audited implementation.

In the whitepaper, **seeding** is private preregistration and **blooming** is the shared public registration step. Neither is an acronym. A **TULIP** is a Taproot UTXO-Linked Identity Program. A **Tulip identity** is the persistent identity it controls; the **identity key** is the current Taproot internal public key. Technical record names such as `SEED`, `PRIOR_BLOOM`, and hash-domain labels below are inherited illustrative identifiers; their spellings and code points are not final. References to a seed batch mean the private KZG stage, not a second public identity registry. Legacy `Rooted/...` hash-domain labels are retained pending the versioned encoding decisions; the naming change does not alter the proposed ID derivation.

The previous detailed Markdown and PDF are archived intact. This document reorganizes that material by implementation concern. The shorter whitepaper changes its presentation, not the agreed registration, authorization, ordering, or reservation rules.

1. Public registration

1.1 Public registration and optional earlier provenance

All public identity registrations expose the same mandatory genesis core and descriptor context. A participant may start here or arrive from seed batch. Public registration is authenticated through a MuSig2 batch or solo submission. The common record semantics must not depend on whether a coordinator was used.

Each entry declares one of two provenance modes:

Mode	Evidence at initial public registration	Historical claim
FRESH	Descriptor and valid public-registration authorization.	Public registration anchor only.
PRIOR_BLOOM	The same descriptor, signed seed record, index, valid opening, and authenticated batch reference, plus public-registration authorization.	Earlier verified seed batch provenance and the public registration anchor.

These mode names are explanatory, not assigned wire codes. Prior evidence must describe the same genesis identity. It must be available to independent verifiers and bound by the public registration's authorization. An unexplained digest or a commitment to unavailable proof data is not sufficient evidence for an indexer to accept the older provenance.

An entry published without prior seed batch evidence is fresh. A participant cannot later attach an omitted proof to upgrade its enrollment date, reset its state, or replace another accepted identity. The public registration anchor and an attached seed batch anchor are different facts with different purposes.

The first valid public registration of the internal key wins. For example, Alice has a seed batch anchor at block 100 but attempts public registration at block 300. An attacker who compromises her identity key and validly registers a different policy at block 250 wins that key's public registration. Alice's earlier private proof cannot override it.

1.2 MuSig2 batches and solo submission

A MuSig2 coordinator prepares one public manifest with an ordered list of genesis descriptors, per-entry provenance modes and evidence bindings, chain and session context, validity conditions, and funding purpose. Participants and the coordinator jointly sign this common message using BIP 327.[17] MuSig2 combines interactive consent to one message; it does not aggregate existing signatures over different seed records.

All required signers review the manifest, including their complete policy, provenance choice, and proposed funding terms. The aggregate key authenticates registration consent only. It does not control later individual TULIPs unless deliberately included in their policies. The manifest must bind optional seed batch evidence per entrant so that it cannot be silently added, removed, or substituted.

A solo participant publishes the same semantic public entry with a protocol-specific BIP 340 consent signature, without a coordinator or MuSig2 signer list. Someone with a prior signed seed batch record but a lost identity secret must retain a route through their committed independent authority. The solo policy-authorized publication and combined recovery profile still needs exact encoding; a fresh signature from the lost genesis key cannot be a universal gate.

Simply constructing a Taproot output using someone else's public key and an attacker-selected tree does not authorize registration. The normal entry profile requires that identity key's consent to the complete descriptor. A valid script-path spend under a spoofed tree is not a substitute for this consent. Prior signed enrollment plus a qualifying committed policy path is a separate profile, not permission to invent an unsigned genesis policy.

1.3 Funding now or later

Public registration does not require an individual UTXO. The proposed MuSig2 purposes are **REGISTER_ONLY** and **REGISTER_AND_FUND**. The second requires one uniquely matching Taproot output per participant and one shared 32-byte commitment to their amounts and scripts. These purpose names replace draft 0.4's funding-dependent activation terminology; exact code points remain open.

Immediate outputs are matched by the script derived from each $P : M$, not wallet labels. The output commitment binds matched outputs in manifest order. The manifest need not repeat each output's index, amount, and script because those data already appear in the transaction. Section 5.5 describes the proposed matching rules. Funding inputs require their own valid Bitcoin signatures; detached manifest consent does not authorize unrelated inputs.

After registration, anyone may fund the current policy. Funding alone changes neither identity authority nor its epoch. The identity may have no UTXO, one UTXO, or many. A UTXO under the current policy is required for key-changing actions, revocation, and the native WoT witness actions discussed below.

Ordinary spending can withdraw donations without emitting an identity action, even if every current output is consumed.

Early funding also has a reputation use case without any planned identity action. An owner may expose funds as a public bounty or honeypot under chosen spending conditions, or make value publicly associated with the identity as visible "skin in the game." Services may weigh that inspectable exposure alongside history and WoT. Funding a bounty does not itself rotate, recover, revoke, or amend WoT. Exact bounty branches, claim conditions, and any honeypot slots are extension specifications, not properties automatically supplied by every funded TULIP. Unless the chosen policy constrains withdrawal, the owner can still move those funds through an ordinary spend.

1.4 Public registration and Bitcoin validation

Bitcoin validates spending rules. **Tuliproot verifiers validate public registration evidence and identity transitions.** Bitcoin consensus does not verify KZG openings, recognize identity IDs, or enforce the application's permanent key registry.[2][6]

Once published, the evidence can be verified without continuing coordinator cooperation. Observers still need authenticated historical transaction and witness data; a pruned node need not retain them.[16] Participants should retain their own copies. Publication makes the disclosed identity and provenance visible and should be a deliberate choice.

2. Identity state and permanent key reservations

An accepted identity is identified by its stable genesis-derived ID. Each accepted policy is an **epoch**, beginning at zero. A state records the current key, tree root, predecessor, and whether the identity is live or revoked. Funding status is separate: a registered identity with no outputs is still registered.

Operation	Effect
REGISTER	Accept the first valid public genesis entry for an unused identity key.
ROTATE	Exercise the committed rotation policy and advance to its allowed precommitted successor.
RECOVER	Exercise recovery authority and publish a fresh successor public key and policy through an authenticated payload, including OP_RETURN data.
REVOKE	End the identity permanently, with no successor.
WoT update	A higher-layer action under the current policy; does not itself rotate or revoke the identity.
Ordinary spend	Move bitcoin without changing identity state.

REGISTER is terminology for the proposed state machine, not a finalized action byte. Registration may accompany a qualifying policy spend and a transition if all evidence and authorization rules are satisfied atomically. The format must not require access to a lost everyday key when independent committed authority already permits the action.

First valid action wins

Actions are processed by confirmed block height, transaction position, and canonical action position within a transaction. For conflicting state-ending actions against one epoch, the first valid action wins. Later actions against that superseded epoch have no effect. A transaction containing conflicting recovery and revocation actions is not rejected merely for that conflict; canonical ordering resolves it. The wire format must define action positions unambiguously.

If recovery confirms before a competing revocation under the old policy, recovery advances the epoch and the stale revocation has no effect. If revocation confirms first, the identity is dead. Recovery cannot restore a revoked identity. A later revocation properly authorized under a newly accepted policy is still valid; recovery does not make the identity permanently immune to revocation.

Comparing two events establishes their chain order, but not that either is valid or that no earlier transition superseded both. Indexers must replay sufficient accepted history. Mempool arrival order has no authority. Reorganizations roll back state and key reservations derived from disconnected transactions, then replay the replacement chain. Permanent reservation means permanent within the accepted chain history, not immunity from chain reorganization.

Old outputs and reserved keys

Alice may have three outputs under one policy. A valid recovery using one advances the identity. The other outputs remain spendable bitcoin, but later spending under their old policy cannot change the successor identity. If all outputs are spent normally before recovery, a new output can be funded under the still-current policy.

Every publicly accepted genesis or successor key is permanently recorded as belonging to its identity. It cannot be reused for another identity or recycled as a new accepted successor. A rotation into an already registered identity key is rejected. Accidental reuse of a secret across wallets or historical setups is a security problem; the affected owner needs a fresh-key recovery where the current policy permits it. This does not grant an exception for a new duplicate registration. Mapping or migration of inconsistent legacy histories is outside this initial profile.

Unknown actions, malformed protocol data, and ordinary key-path spends cause no identity transition. A syntactically invalid candidate does not consume the opportunity for a later valid action. Signature, current-state, and successor checks precede every state change.

3. TULIP policies and action authentication

The participant chooses how to authorize ROTATE, RECOVER, and REVOKE. A rotation may use a finite precommitted succession plan. Recovery selects a fresh key under the authority committed for recovery. The fresh public key, never its secret, is carried in the authenticated recovery payload. Compromise of recovery authority is considered compromise of the account itself.

Both rotation and recovery can be needed after loss of the current identity key. The protocol therefore supports independently authorized transitions without a fresh signature from that key. A policy that requires it cannot claim lost-key recovery unless usable authorization was prepared beforehand, for example with appropriately constructed presigned transactions. Presigning must account for the actual inputs, payload, outputs, and signature modes; it is not a generic reusable authorization for arbitrary future UTXOs.

The following are recommendations, not universal authorization requirements:

- Use authority independent of the everyday/internal identity key for rotation, recovery, and revocation.
- Use different authorization keys for rotation and recovery. Separate offline keys, k-of-n script policies, and carefully managed guardian arrangements are options.
- A separate single authorization key may be backed up with Shamir secret sharing. This differs from threshold signing: reconstructing a key and producing a distributed signature have different exposure and implementation properties.
- Require a signature from the new successor identity key, bound to the particular transition, to confirm control before changing state. This does not require the old lost identity key. Successor acceptance remains a policy choice.
- Keep recovery material, scripts, control paths, guardian contacts, and enrollment receipts backed up and usable. Knowing a guardian's public key does not guarantee future assistance or compatible software.

Full disclosure is the threat model

Assume an attacker can know the entire script tree and control the everyday identity secret. Security must rest on the policy's independent authorization, not hidden branches or secrecy of m . Repeated script-path spends can expose more branches. Private submission may reduce public preconfirmation exposure but is not a security assumption.

The everyday identity key is also the Taproot internal key. With the tweak available, its holder can spend funds through the key path; script conditions cannot restrict that otherwise valid spend.[2] Such spending does not automatically authorize identity transitions. A public-key-only attacker can construct another output containing an attacker-controlled script tree, but cannot supply the victim's required descriptor consent. If the identity secret is compromised before public registration, the first-public-registration rule provides the boundary described in Section 1.

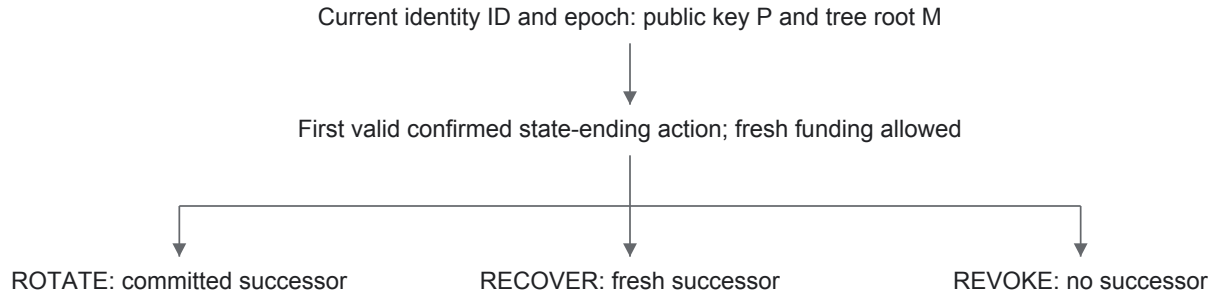
An identity-key-only fallback for state changes is highly discouraged. A user can choose a delayed fallback, but its security claim must acknowledge that a compromised key may eventually satisfy it. A relative timelock on an old output can already have matured before compromise; it does not automatically create a new response window when a rotation is announced.[20] Exact delay semantics belong in the chosen policy and specification.

Binding the action

Recognized action paths must bind action type and policy context. Relevant signatures must authenticate the identity ID, epoch, predecessor, and complete dynamic payload, including successor data. Payloads may be placed in OP_RETURN outputs covered by those signatures. Dropped, unsigned witness text is not equivalent. A precommitted successor must match the committed target according to the recognized template and verifier rules.

This is not a claim that ordinary Bitcoin Script introspects arbitrary OP_RETURN contents or provides a covenant. Bitcoin enforces the script and signature rules; the metaprotocol additionally checks the successor and transaction relationship. Concrete templates and exact signed encodings remain implementation work.[6]

Figure 1. Policies persist; outputs provide action opportunities



Rotation and recovery preserve the ID and advance the epoch.

Old outputs remain coins; their later spends cannot change the identity.

Prior commitments are not transitions. Future roots must be constructed without circular dependencies. Superseded outputs retain spending value but no identity-state authority.

Recovery after funds have been spent

Draining the current outputs does not erase the policy. A defender can fund it again and attempt recovery, but an attacker with key-path spending authority can race the new output. Section 9 sketches experimental fee-funded rotation and recovery packages. It may limit a failed attempt's exposed value under its assumptions; it does not guarantee confirmation, favorable miner behavior, or a universal attacker-to-defender cost ratio. Actual transactions and relay behavior must be tested before making economic security claims.

4. Verifier requirements

A verifier separates three questions: what evidence existed, who is authorized now, and what chain history has been accepted.

Claim	Required checks
Public registration	Complete descriptor and derived ID; declared provenance mode; solo or MuSig2 authorization; confirmed carrier; earlier accepted key reservations.
Prior seed batch provenance	Signed seed record and matching descriptor; coordinator declaration; count and index; pinned parameters; valid opening; authenticated earlier anchor; inclusion in the initial public entry.
Current identity state	Accepted ordered action history through a sufficiently recent chain view; current epoch and predecessor; policy and successor validation; terminal revocation and permanent key reservations.
Present authentication	Fresh challenge from the currently authorized key, bound to service, identity, epoch, action, and nonce.

An enrollment signature is not a fresh login proof. An old receipt can remain historically valid after rotation or revocation without granting present authority. Likewise, two event locators or a proof of inclusion do not establish a complete current history.

Verifiers need a validating chain view or appropriate authentication of both transaction and witness evidence. A transaction ID is a locator, not authentication of arbitrary witness data.[7] An indexer is useful

for discovery, but trusting its unverified state answer also trusts its omissions. Header, witness, and history requirements must be explicit in any light-client profile.

Coordinators can delay service, withhold receipts, refuse a batch, or make their own attestations. They cannot replace an accepted participant policy merely by publishing a record. MuSig2 requires cooperation during the initial session; it grants no later aggregate authority over individual TULIPs. A valid receipt plus policy backups should permit solo public entry through an appropriate authorization profile. Loss of the initial key must not make MuSig2 participation the only path to recovery.

Section 5.6 outlines verifier state and event processing. Its ordering rules are agreed design choices, while serialization, exact templates, and failure granularity remain to be finalized.

5. Cryptographic construction and verifier processing

This appendix defines the intended checks and sketches records. `Encode`, record names, field names, and purpose labels are explanatory until a versioned wire specification selects their exact representation.

5.1 Genesis descriptor and stable identifier

Let P_0 be the initial x-only identity key, M_0 the initial Taproot tree root, and u a random public identity nonce. Build scripts with `chain`, u , and epoch context, then derive:

```
ID = TaggedHash("Rooted/ID",  
    Encode(genesis_version, chain_id, P_0, M_0, u))
```

The participant constructs or verifies the descriptor. `Encode` must be canonical and unambiguous. Scripts used to construct M_0 cannot depend on the final ID, directly or indirectly, because that creates a circular definition. Future precommitted roots must also form an acyclic construction. The ID can appear in authenticated transaction payloads checked by verifiers.

Changing a genesis field changes the candidate ID; it does not permit a second accepted identity using an already reserved key. The public registry, not the ID hash alone, enforces uniqueness. A publicly accepted successor preserves the original ID while updating the current key and root.

Taproot derivation follows BIP 341, including even-y interpretation of x-only internal keys, tweak validity, and control-path/output-parity checks.[2] An absent script root and a zero-filled 32-byte root are different inputs. Any empty-tree profile must specify an explicit representation.

5.2 Signed seed record and common public entry

A seed batch record binds at least:

```
protocol_version, chain_id, genesis_descriptor, identity_id,  
coordinator_key, batch_id, coordinator_challenge,  
delegation_scope, enrollment_nonce, validity_conditions
```

```
sig_i = BIP340.Sign(sk_i,  
    TaggedHash("SEED/consent", Encode(record_i)))  
SEED_i = Encode(record_i, sig_i)  
y_i = HashToField_Fr("BLOOM/leaf", SEED_i)
```

Hash-to-field targets BLS12-381's scalar field with a selected construction and negligible statistical bias. An arbitrary public key or hash is not simply treated as a canonical scalar. RFC 9380 provides relevant methods; the exact scalar-field profile remains to be selected.[11]

The shared public entry conceptually binds:

```
genesis_descriptor,  
provenance_mode = FRESH | PRIOR_BLOOM,  
prior_evidence = absent | (SEED, index, opening, batch_reference),  
validity_conditions
```

Common context may be inherited from a canonical header. Verifiers recompute the ID, confirm matching descriptors, and validate the declared evidence mode. All evidence needed to accept PRIOR_BLOOM must be published or unambiguously resolvable as authenticated chain data. Later attachment of omitted evidence is not accepted.

For the normal solo profile, the identity key signs a domain-separated hash of the complete public record, including chain, version, nonce/session context, purpose, entry, and any immediate funding commitment. A lost-key solo profile must instead combine previously valid signed enrollment evidence with a qualifying committed policy action. Its exact serialization and consent checks remain open; Bitcoin spend validity alone cannot authenticate a newly invented descriptor.

5.3 Indexed KZG membership

For a fixed evaluation domain of size N , encode each enrollment as the field value y_i at the specified point w_i . Pad unused positions using a separately defined padding rule. Interpolate a polynomial f of degree less than N :

```
f(w_i) = y_i  
C       = Commit_SRS(f)  
pi_i    = Open_SRS(f, w_i)
```

The authenticated declaration binds the scheme, parameter identifier, domain size, actual record count, batch context, and c . A verifier checks the index against the record count, derives w_i , reconstructs y_i , and verifies the opening under the pinned parameters. Curve encodings, subgroup membership, and permitted identity elements require explicit validation rules.[4][5]

An evaluation proof at an arbitrary point is not sufficient to define indexed membership. Ordering, domain selection, padding, and count are part of the protocol. An index belongs to its batch. A fixed evaluation domain does not require the same identity to occupy the same position in other batches. Changing positions also does not hide a disclosed identity ID or public key.

Using one record per evaluation and ceremony parameter set #4, the maximum is 32,768 positions. Smaller batches can use specified supported domains. No fixed total receipt size is claimed until the full wire format is defined.

5.4 Action authentication and replay protection

A recognized action binds the chain, identity ID, current epoch, predecessor, action type, and dynamic successor or higher-layer payload. The verifier checks the spent policy against the accepted current (P, M) and checks that its recognized path grants that action. A valid Bitcoin spend with an unrelated label is not sufficient authorization.

For the two-preimage ROTATE construction, the author specifies authorization as the ability to make a valid spend of a current TULIP UTXO with the required two preimages and the proper ROTATE witness header. The preimages must satisfy the selected rotation path's commitments. One reveals the successor's internal public key. No additional detached protocol signature is universally required: signatures, thresholds,

or other spending conditions are those of the chosen TULIP. The contents of the second preimage and the exact header and witness encoding remain to be specified; it is not assumed here to be a tree root or a secret authorization token.

Precommitted witness values are constrained by their script commitments. Where signatures authorize dynamic action data, they must cover the action-bearing outputs or an equivalently specified authenticated payload. Arbitrary OP_RETURN text or witness material merely dropped by a script does not acquire authority to select a different successor. The final templates must define the proper header, its recognized action path, the two preimage checks, and how the complete successor policy is resolved. They must not assume an unspecified covenant or transaction-introspection opcode.

ROTATE and RECOVER preserve ID, advance epoch by exactly one, and reference the currently accepted predecessor. A successor key must be fresh under the permanent accepted-key history; successor root and key encodings must be valid. REVOKE is terminal. A policy may additionally require successor acceptance signed by the new key, bound to this ID, epoch, predecessor, action, and successor policy. It is recommended, not universally required.

Preimage material must not be treated as fresh authority merely because it appears again. New epochs need new policy context. Presigned constructions, hashlocks, signature thresholds, and backup schemes must be specified separately enough to demonstrate the claimed loss and compromise properties.

5.5 Common MuSig2 public-batch profile

The manifest binds protocol version, chain, batch/session identifier, coordinator key, ordered public entries, validity conditions, and shared purpose. Every entry contains $P : M$ and the rest of its descriptor; the derived ID need not be redundantly serialized. Prior seed batch evidence is bound per entry. No participant may sign only an unexplained digest.

Use one of the conceptual purposes REGISTER_ONLY or REGISTER_AND_FUND. The first reserves identities without requiring individual outputs. The second adds one shared 32-byte `output_commitment`. Require distinct derived participant scripts and exactly one output matching each script in the public anchor transaction. Collect the matched outputs in manifest entry order:

```
required_output_i = unique output matching Taproot(P_i, M_i)
output_commitment = TaggedHash("Rooted/public-outputs",
    Encode(required_output_1, ..., required_output_n))
```

Each encoded output includes its actual amount and script. Unmatched outputs, including change or carrier outputs, are excluded. An extra matching output makes the immediate-funding batch ambiguous and invalid under this proposed profile. Participants review the output vector before signing. The rule applies to onboarding only; later funding may create many outputs.

The signer list contains the distinct participant initial keys and the coordinator key in prescribed order. This profile rejects duplicate signer keys. BIP 327 accepts compressed public keys: normalize each x-only identity key to its even-y representative, encode it as `0x02 || P`, and normalize its signing secret accordingly.[17] Follow all BIP 327 nonce and session rules. The coordinator must not substitute the signer list or reuse signing nonces across sessions.

```
msg = TaggedHash("Rooted/public-musig2", Encode(manifest))
agg_key = MuSig2.KeyAgg(ordered_normalized_signer_keys)
joint_sig = MuSig2.SignTogether(signers, msg)
```

`SignTogether` denotes the BIP 327 interaction, not a new signing algorithm. Verifiers derive the aggregate key from the full signer list and check the signature, each declared evidence mode, the registry, and any funding conditions. A participant whose key has been lost cannot be required to join this genesis-key signing profile to exercise independent committed recovery; that participant uses the separately specified solo policy-action path.

The manifest excludes its own signature and final transaction ID. A replay cannot create a duplicate registration, reset an epoch, or revive a revoked identity. The detached joint signature authorizes the manifest, not unrelated funding inputs. This construction is not cross-input signature aggregation.

5.6 Proposed verifier state and processing rules

Maintain at least the following chain-derived data:

Store	Contents
Identity state	ID, genesis descriptor, public anchor, immutable accepted provenance, current epoch/key/root, predecessor, live or revoked status.
Reserved keys	Every accepted identity key, its owning ID, and first accepted position; retired and revoked keys remain recorded.
Chain undo data	Previous states and reservations needed to undo disconnected blocks.
Evidence cache	Authenticated declarations, manifests, witnesses, and proofs or resolvable references needed to reproduce validation.

Process transactions in confirmed order. A canonical parser must first establish event positions and evidence coverage. For a MuSig2 batch, authenticate the complete manifest and signer set before trusting entries. Entry-level versus whole-batch failure handling is still a wire-profile decision; no implementation may assume a signature authenticates entries omitted from its message.

For a public registration candidate:

1. Validate version, chain, canonical encoding, public authorization, descriptor, and derived ID.
2. Validate declared provenance. `PRIOR_BLOOM` requires the matching signed record, authenticated earlier declaration, in-range index, pinned parameters, and valid opening. `FRESH` carries no claim to earlier seed batch age.
3. Check that the identity ID has not already registered and the initial key has never been accepted or reserved. Check any immediate-funding requirements.
4. Accept the identity at epoch zero, fix its provenance, and reserve the initial key. Apply a combined policy action atomically only if its complete profile is valid.

For a state-ending action candidate:

1. Resolve ID, require live status, and check current epoch and predecessor.
2. Verify the spent output matches the current policy and the recognized action path authorizes the complete payload. Apply the TULIP's selected signature, threshold, preimage, and timing requirements.
3. For a successor, verify key/root validity, freshness against accepted key history, allowed target constraints, and any policy-required successor acceptance or output relationship.

4. Apply ROTATE or RECOVER to the next epoch and reserve the new key, or mark REVOKE terminal. Later candidates against the old epoch fail the current-state check.

For native WoT, validate its current-policy spend and authenticated record before passing it to the higher-layer indexer. It does not itself end the identity epoch. Ordinary spending has no identity effect. A complete WoT ordering and update profile is separate work.

The parser must define multiple actions per input, output, or witness, and their canonical order, before these rules become interoperable. This pseudocode is a specification outline, not a claim that these decisions are already implemented or tested.

6. Weighted WoT and other higher layers

6.1 A metaprotocol above the identity metaprotocol

Weighted web of trust is the first proposed higher layer above Tuliproot. An identity publicly expresses whom it vouches for. These are directed, unilateral statements: the target need not approve, countersign, or reciprocate. Observers verify the source's authority. Recipient consent and source authorization are separate questions.

An identity needs a current-policy UTXO to perform a native WoT witness action. A WoT record may use OP_RETURN data together with an authenticated witness reveal. The witness signing keys may also bootstrap a WoT set, under explicitly specified role and weighting rules. Recovery guardians must not silently become maximum-weight endorsements merely because they are guardians; a recognized WoT declaration or template must give the keys that meaning.

WoT indexers interpret this higher layer. The proposed default follows the vouched-for identity through valid rotation and recovery. Revocation makes the connection dead. Historical statements remain observable without constituting live endorsements. Permanent identity-key reservations support mapping listed historical pubkeys to one identity ID. Handling a vouch made before the target is publicly registered needs explicit indexer rules.

6.2 Compact grouped updates

The design target is approximately **two control bytes per group of pubkeys**. One shared weight applies to every key in that group. Groups can be concatenated in an OP_RETURN and can express additions, amendments, and replacement of the entire set. Weight **0 removes** the listed keys; it does not retain a zero-trust entry. A weight change can amend an existing entry. Exact operation and replacement boundaries remain unspecified.

Weights range from 0 to 16. Weight 8 is the proposed default for a headerless pubkey list; weight 0 removes the listed keys. Higher weights signal a stronger vouch and allow finer distinctions. No nonzero weight is assigned a particular relationship or reason by the protocol. Applications consuming Tuliproot WoT data determine how to interpret these signals and use them in their own models.

A direct encoding of 17 weight values requires five bits. The rest of a two-byte header must encode operation and bounded count information. This paper does not assign a final bit layout, count limit, byte order, or headerless discriminator. Different weights require different groups. Two bytes do not include the pubkeys themselves. Parsing must unambiguously distinguish default lists from headed groups and define duplicate keys and sequential updates.

6.3 Private targets and extensions

An external party may vouch for a pubkey whose seed batch receipt and tree root remain undisclosed. The target needs neither public registration nor its own UTXO merely to receive that vouch. This may help private accounts interface with services. Publishing a pubkey in an onchain WoT record reveals that key; it does not by itself reveal the target's receipt, tree root, or cohort membership. Privately showing a receipt to a service is a different disclosure choice.

Other proposed uses include linking an identity to a honeypot with precommitted slots, relating it to an inscription reveal, or declaring an associated asset collection. These require separate record and verification semantics. A link alone does not establish ownership, spending authority, or endorsement. Withdrawing bitcoin donated to a public TULIP remains ordinary spending, not a metadata action.

6.4 Limits of provenance and compatibility

Age proves an old signed commitment, not continuous control or good behavior. Funds indicate value under spending rules, not personal identity. An untouched honeypot does not establish secrecy of the key. Applications may assess coordinators and attestations, but attribution is not a guarantee of conduct.

Disclosure links identity keys, policies, coordinators, and historical anchors. Ordinary KZG openings do not guarantee anonymity. Script-path spending exposes the exercised branch and control path, not unused scripts; revealing the descriptor already links its internal key to its Taproot address.[2] MuSig2 public manifests disclose their descriptors and cohorts regardless of funding.

Nostr uses secp256k1/BIP 340 keys, but clients need additional support to follow key changes and construct TULIP transactions. NIP-07 provides event signing, not a general Bitcoin transaction-signing interface.[14][15] A future NIP should define stable-ID references, current-key resolution, revocation behavior, and client verification. This document does not claim that existing clients implement those features.

7. KZG tradeoffs and setup assumptions

A Merkle tree is a simpler alternative. Its root commits to many records, and an inclusion proof supplies the sibling hashes needed to reconstruct that root. It avoids a KZG trusted setup and pairing verification.[8]

The comparison here holds the enrollment protocol constant. Both options keep signed seed records offchain, publish an authenticated batch commitment, deliver individual receipts, and have applications verify enrollment. Neither requires publication of the complete batch. Neither commitment can reconstruct missing records or proofs by itself.

KZG becomes more attractive when participants publish many individual proofs onchain. Its opening has constant size, while a conventional binary Merkle proof grows with the tree's depth.

Construction	Commitment	Proof for 1,024 positions	Proof for 32,768 positions
Binary Merkle tree with 32-byte hashes	32 bytes	320 bytes	480 bytes
KZG with the proposed BLS12-381 encoding	48 bytes	48 bytes	48 bytes

These figures count the commitment and cryptographic proof components only. They exclude the seed record, signatures, index, declaration, chain reference, transaction overhead, and any auxiliary evidence.

The Merkle figures are ten or fifteen sibling hashes for full binary trees; the KZG figure follows the selected encoding.[5][8]

At 32,768 positions, KZG saves 432 raw proof bytes per disclosed record compared with that Merkle construction. Its batch commitment is 16 bytes larger. The practical fee difference depends on how the data is carried and on the complete transaction. This version uses one indexed opening per enrollment and makes no cross-batch aggregation claim.

The tradeoff also includes different security assumptions. Merkle membership relies on a secure hash construction. KZG additionally relies on pairing-based cryptography and the setup's secret randomness, often called its trapdoor, remaining unknown. Ethereum's ceremony is designed to preserve that secrecy if at least one participant contributes honest randomness and erases it.[4][10]

If the trapdoor becomes known, an attacker can create false openings against earlier commitments. For example, the attacker could sign a new seed record with their own key and make it appear enrolled in an old seed batch. The signature would be valid, but the claimed historical inclusion and coordinator attribution would be false. A verifier relying only on the commitment and opening could not detect the substitution. This breaks enrollment binding; it does not forge Schnorr signatures, alter confirmed Bitcoin transactions, or satisfy another identity's recovery conditions.[4]

The proposed profile uses Ethereum ceremony parameter set #4, with 32,768 G1 powers supporting polynomials of degree at most 32,767.[9] With one record per evaluation, this permits at most 32,768 positions per batch. This is a limit of the selected cryptographic profile, not a Bitcoin consensus limit. Larger populations use additional batches or a separately specified profile. A deployment must pin the exact verified parameters, their digest, and encoding.

Using these public parameters requires neither Ethereum transactions nor continuing permission from their publisher. Standard Ethereum blob APIs also do not automatically implement this larger enrollment vector. This proposal selects KZG for compact public disclosure; a Merkle profile remains a simpler alternative when avoiding setup dependence matters more than proof size.

8. Byte accounting and transaction costs

Batching shares costs at different stages. Optional seed batch preregistration places one commitment onchain and keeps signed records private. Shared public registration publishes descriptors through MuSig2 or solo submission and adds prior seed batch evidence for entrants who claim it. Creating outputs is a separate, optional immediate cost. These stages must be added together when estimating a participant's full lifecycle.

8.1 Fixed components and growing records

The following sizes are exact for the stated encodings. A BIP 340 signature is 64 bytes; KZG commitments and individual openings use 48 bytes in this profile.[3][5] A Taproot output has a 34-byte script: one version opcode, one push opcode, and the 32-byte output key. Its complete transaction-output serialization is **43 bytes**: an 8-byte amount, a 1-byte script length, and that script.[2][18]

Bitcoin charges transaction space by weight. A non-witness byte contributes four weight units (WU); an additional witness byte contributes one. The virtual size is the complete transaction weight divided by four, rounded upward.[7]

Component	Serialized size	Weight contribution
Identity core $P : M$, in witness	64 bytes	64 WU, equivalent to 16 vB
One joint consent signature, in witness	64 bytes per batch	64 WU, equivalent to 16 vB
KZG commitment or one opening, in witness	48 bytes	48 WU, equivalent to 12 vB
One newly created Taproot output	43 non-witness bytes	172 WU, equivalent to 43 vB

These are components, not complete transactions. The 43 bytes measure an output in a transaction, not its full database storage or its later spending cost.

Let n be public-manifest participants and m the additional encoded bytes per MuSig2 entry beyond its 64-byte core. This includes the identity nonce, provenance-mode framing, and other per-entry metadata; optional seed batch evidence is counted separately below. Chain and batch context can be stored once where shared. Identity IDs are derived rather than repeated in the proposed compact MuSig2 manifest. The nonce and remaining context are still required; m is not assumed to be zero.

Let H_B be the seed batch declaration bytes other than its commitment, including coordinator authentication. Let H_M be the common MuSig2 header bytes, including the coordinator key and purpose, but excluding participant entries, the joint signature, and the immediate-mode output commitment. Their exact lengths depend on the final encoding. No extra coordinator signature is added to MuSig2: the coordinator participates in the joint signature.

Enrollment measure	seed batch	Public MuSig2: register only	Public MuSig2: with outputs
Shared application payload	$H_B + 48$	$H_M + 64$	$H_M + 96$
Added application payload per participant	0 within the batch	$64 + m$ bytes	$64 + m$ bytes
Required new participant outputs	None	None	One each: 43 bytes
Added core-and-output weight, before metadata and framing	0	64 WU / 16 vB	236 WU / 59 vB
Individual descriptors public at enrollment	No	Yes	Yes

This table counts base components before optional per-entry seed batch evidence, defined as E_i below. The immediate-mode shared payload includes its 64-byte joint signature and one 32-byte output commitment. The per-participant core remains $P : M$ in every mode. A count field can grow at an encoding boundary. The private seed batch stage's zero marginal record payload means adding a participant **before sealing the batch**, within its capacity; an anchored commitment cannot be appended to for free.

8.2 Onchain growth and fees

Let E_i be the additional witness-carried prior seed batch evidence for public entry i , including its signed seed record, opening, index, and reference, or zero for a fresh entry. Let $E = \sum(E_i)$. Shared descriptor fields may be deduplicated only after a canonical encoding defines that optimization. For witness-carried application data, component weights are:

$$W_B(n) = F_B(n) + H_B + 48$$

$$W_M(n) = F_M(n) + H_M + 64 + n*(64 + m) + E$$

$$W_{Mplus}(n) = F_{Mplus}(n) + H_M + 96 + n*(236 + m) + E$$

$$vsize = \text{ceil}(W / 4)$$

$$\text{fee in satoshis} = \text{ceil}(\text{feerate in sat/vB} * vsize)$$

F is the remaining transaction weight: funding inputs and their authorizations, carrier script, control block, witness framing, count fields, change, and other outputs. It can grow with the batch, so it is not a universal fixed constant. If metadata lengths differ, replace $n*m$ with their sum. The 236 is $64 + 172$: the witness core plus one actual Taproot output. Moving application data outside witness changes its weight contribution.

For example, these are the **known component subtotals for 1,000 participants**, excluding H , all per-entry metadata, optional seed batch evidence E , and F :

Route	Component bytes	Component weight / equivalent vB
seed batch: one commitment	48	48 WU / 12 vB
MuSig2: cores and joint signature	64,064	64,064 WU / 16,016 vB
MuSig2: those components, output commitment, and 1,000 outputs	107,096	236,096 WU / 59,024 vB

These subtotals illustrate the slopes; they are not usable transaction-size or fee quotes. Each extra byte of per-entry metadata adds 1,000 witness bytes to this example. A commit-and-reveal carrier also requires counting the earlier commit transaction when estimating the complete enrollment cost.

Length and count encodings matter. For example, increasing the total output count from 252 to 253 expands its CompactSize field by two non-witness bytes, adding 8 WU beyond the new output itself.[18] Carrier push instructions and witness-item lengths also consume space.

8.3 Offchain inclusion evidence

Let a complete signed seed record occupy $s_i = 128 + e_i$ bytes: the 64-byte identity core, its 64-byte consent signature, and e_i bytes of remaining signed context and record encoding. Let I be the encoded batch-index length. The verifier derives the KZG field value from the seed record and its evaluation point from the index and domain; neither needs a separately transmitted 32-byte value.

Evidence	Bytes needed, before chain evidence	Growth with batch size
KZG opening alone	48	Constant
seed batch enrollment, with an authenticated declaration already available	$s_i + I + 48$, or $176 + e_i + I$	Independent of other participants, apart from index encoding
Public MuSig2 enrollment, without a cached manifest	$H_M + 64 + n*(64 + m) + E$; add 32 in immediate mode	Linear in participants, plus optional prior evidence

The **176-byte subtotal is not a complete receipt**. A usable receipt must also include the authenticated declaration or a resolvable reference. A self-contained proof must include the declaration and necessary anchor evidence. A two-byte index is sufficient for 32,768 positions, but the final format must define its encoding. A single claimed total such as 244 bytes would be premature while these fields remain unspecified.

MuSig2 alone supplies no small individual membership opening. The verifier needs the complete signed manifest and ordered signer list to establish whose consent the aggregate key represents.[17] Once that public data has been fetched and verified, later presentations can use a short reference. This is a caching benefit, not a constant-size independent proof. Immediate funding also requires verified evidence of the matched outputs; the manifest and signature alone prove consent, not their confirmation.

All entries above concern enrollment. Proving current authority additionally requires a fresh challenge where appropriate and a sufficiently current view of rotations, recoveries, and revocations. Chain headers, transaction inclusion evidence, and witness authentication depend on the verifier's chain access; they are not included in these enrollment subtotals.

8.4 Prior evidence at public registration and later actions

For a participant presenting prior seed batch evidence at initial public registration, let R be the batch-reference bytes. An unoptimized evidence bundle contributes:

$$\begin{aligned} E_i &= s_i + 48 + I + R \\ &= 176 + e_i + I + R \end{aligned}$$

This includes a complete signed seed record, even though its descriptor also appears in the public entry. A final shared encoding may remove that duplication without omitting any authenticated fields. Public registration consent, joint or solo, is counted separately. A policy-spend authorization has its own input, witness, script, and control-path costs.

If this evidence is carried in witness, each byte contributes one WU before framing. Data in an `OP_RETURN` output is non-witness and has a different weight cost. The earlier declaration can be retrieved through its authenticated reference, but the opening alone is not a complete provenance claim.

Prior evidence cannot be added after registration to upgrade age. Later rotation, recovery, and WoT actions reference the existing public registration and accepted state; they do not republish the original manifest. Creating a TULIP output adds 43 non-witness bytes wherever it occurs. Spending an existing output still adds an input and its authorization. Successor outputs, recovery OP_RETURN data, and WoT records must be counted separately.

8.5 Capacity and lifetime costs

The selected KZG setup supports at most 32,768 positions in one seed batch. Enrolling T participants therefore needs at least $\text{ceil}(T / 32768)$ batches under this profile. More participants mean more offchain records and proof-generation work, even while one batch's commitment remains 48 bytes.[9]

MuSig2 also requires offchain coordination. Each signer's public nonce and partial signature contribute 66 and 32 bytes respectively; distributing the full manifest, session data, replies, and retries is additional.[17] Constant signature size does not imply constant signing communication.

Large public manifests must fit actual transaction and carrier limits. The referenced Bitcoin Core policy limits standard transactions to 400,000 WU; Bitcoin's consensus block limit is 4,000,000 WU.[7][19] A carrier must also satisfy applicable tapscript and relay rules.[6][19] An arbitrary large manifest cannot simply be assumed to fit one witness stack item. Until encoding and funding are implemented, this paper does not claim a maximum MuSig2 participant count per transaction.

seed batch's largest onchain savings occur when many preregistered identities remain private. If all participants eventually disclose, create outputs, and manage their policies, those costs grow with the number who act. A complete comparison must add enrollment anchors, any earlier commit transactions, public registration disclosures, output creation, and later action spends. This keeps the preregistration advantage distinct from lifetime blockchain usage.

9. Experimental fee-funded rotation and recovery

This section proposes a fee-funded defense against an attacker who knows the current internal identity secret and can derive the Taproot tweak, while the selected rotation or recovery authority remains secure. It does not protect an identity whose independent action authority is also compromised. No relay or economic experiment is claimed to have been completed.

9.1 Zero-fee funding parent and fee-paying action child

Use a two-transaction version-3 package:

Transaction	Inputs	Relevant outputs and effect
F: fund	A confirmed input under an uncompromised key.	A small current-policy TULIP output of value V and protected change. F pays zero fee.
R: act	The TULIP output from F, plus a separate confirmed fee input under an uncompromised key.	A valid rotation or recovery spend, required payload and outputs, and protected fee-input change. R pays the package fee.

The fee input, funding input, and protected change must not use the exposed identity secret. Signatures and spending conditions must prevent an attacker from redirecting that fee input or change. Merely splitting funds across inputs controlled by the compromised key does not protect them.

An attacker may construct A, a key-path spend conflicting with R. If F and A confirm while R does not, the fee input used only by R stays unspent. For zero-fee F, no other exposed outputs, and no separately paid defense fees:

```
defender failed-round loss = V  
attacker net funding cost = fee(A) - captured value from V
```

Captured value includes coins stolen into change or used toward A's fee. Misuse of the identity may have separate value to an attacker; that is outside this transaction accounting. A successful action costs the owner its actual confirmation fee. If F pays any fee, or other outputs are exposed, those costs must be added to the failed-round loss.

For a 330-sat Taproot output and a 20,000-sat action fee, a replacement must satisfy the node's fee rules rather than merely be smaller than the script-path spend. A confirmed attacker transaction paying more than 20,000 sats has a net cost above 19,670 sats if it captures all 330 sats. The victim's rejected fee offer is not itself a payment. Five confirmed failed rounds at this example bid expose 1,650 sats to the victim and require attacker fees above 100,000 sats, with net attacker funding above 98,350 sats after crediting the stolen outputs. These are conditional example values, not a guaranteed cost ratio.

9.2 Replacement policy and repeated attempts

If R is already accepted, the referenced Bitcoin Core replacement rules require a conflict to cover the fees of the evicted transactions, pay the incremental relay charge, and strictly improve the mempool feerate diagram, in addition to other policy checks.[13] The shorter key-path witness does not eliminate the absolute-fee requirement. The actual replacement calculation depends on the evicted set and the node's configuration. Limited package replacement also has its own acceptance requirements.[22]

An attacker whose cheaper conflict reaches a miner first is not forced by consensus to replace the owner's bid. Nodes can have different mempool contents and miners can select transactions outside the public replacement policy. Each confirmed blocking spend may force a new funding round, but an unconfirmed RBF exchange does not charge every announced bid. Fee payment happens on confirmation.

Private submission is an optional propagation strategy, not a confidentiality or confirmation guarantee. A recipient miner or relay can see the package. Public competition exposes the offer and permits repeated replacements. Once a legitimate state-ending action is confirmed and accepted under the identity rules, spending an old-policy output cannot undo it.

9.3 TRUC topology and P2A placement

Both F and R use version 3. R has just one unconfirmed parent; its additional fee input is confirmed. Under the referenced TRUC profile, an unconfirmed parent cannot have more than one unconfirmed descendant, and a child with an unconfirmed TRUC ancestor is limited to 1,000 sigop-adjusted vB. The actual policy script, preimages, header, control path, inputs, and outputs must fit that limit.[12]

P2A is a separate, keyless anchor output type; it does not implement identity authorization.[21] F may optionally include a P2A output. R then spends that anchor alongside the TULIP output and its protected fee input, staying within the same parent-child topology. Under ephemeral-dust handling, F must pay zero fee and its child must spend the parent's ephemeral dust output. The candidate can use one zero-value P2A anchor with a 330-sat non-dust TULIP output.[23]

Adding a fee child below R would create an unconfirmed $F \rightarrow R \rightarrow C$ chain and violate this TRUC topology. A different arrangement is possible when the funding output is already confirmed: R can be the zero-fee action parent with a P2A output, and C its fee-paying child. That arrangement must account separately for the fee used to confirm the funding transaction; it does not establish a 330-sat all-in failed-round cost.

P2A is anyone-can-spend. An adversary may attach a competing child spending only the anchor, rather than stealing the TULIP output. TRUC sibling-eviction and replacement behavior must be tested for this case. If an anchor-only child confirms, it has not itself consumed the TULIP output or changed the identity. The owner may still attempt an action using that now-confirmed output. An optional anchor is not the source of the cost asymmetry; protected fee funding and conflicting spends are.

9.4 Output values and validation work

At a dust relay rate of 3,000 sat/kvB, the referenced dust calculation gives 330 sats for the 43-byte Taproot output plus its policy-estimated spending size. A 240-sat P2A threshold applies to the shorter anchor output, not to Taproot. Neither figure is a consensus minimum. A smaller TULIP output may be possible under a separately validated ephemeral-dust arrangement; 240 sats cannot be advertised as an interchangeable standard TULIP minimum.[19][21][24]

The one-zero-anchor example keeps total exposed value V at 330 sats. Using a 240-sat nonzero P2A anchor as well can expose another 240 sats, which must be counted if captured. More funded TULIP outputs or other attack-controlled spending paths can expose additional value. Existing losses from draining the identity are separate from the cost of a new attempt.

Before presenting this as a working defense, pin the Bitcoin Core version and settings and test signed transactions on regtest: package acceptance; child size; protected inputs and change; direct conflicts in both arrival orders; replacements and package feerate checks; P2A-only sibling eviction; ephemeral-dust spending requirements; private/direct submission assumptions; and both successful and failed-round accounting. This is a proposed incentive mechanism, not guaranteed recovery or a universal attacker-to-defender cost ratio.

10. Illustrative transaction layouts

These layouts omit wire bytes, amounts, fee inputs where irrelevant, and full scripts. They illustrate required relationships, not ready-to-broadcast transactions. Carriers must satisfy the selected Bitcoin policy and script constraints. A witness-carried manifest may require a prior commitment transaction; an arbitrary witness item cannot be assumed relayable.

10.1 Fresh public registration, funding deferred

Public transaction:

```
inputs: ordinary authorized funding/carrier inputs
public record: descriptor, FRESH, REGISTER_ONLY, consent
outputs: carrier and change as required; no individual TULIP output
```

Indexer: accept ID at epoch 0; reserve P_0; fix public-only provenance.

Later: a payment creates an output to Taproot(P_0, M_0).

That payment alone emits no identity action.

A solo consent signature or a valid MuSig2 manifest authenticates registration. Lack of an individual UTXO does not invalidate the registration.

10.2 Mixed MuSig2 batch with prior seed batch evidence

Manifest:

```
participant A: descriptor A, PRIOR_BLOOM, SEED/index/opening/reference
participant B: descriptor B, FRESH
purpose: REGISTER_AND_FUND
shared commitment: matched output amounts and scripts
consent: joint MuSig2 signature from A, B, and coordinator
```

Outputs: exactly one match for A; exactly one match for B; other outputs.

Verify common consent, each entry's evidence, unused identity keys, and matched funding. A retains verified prior age; B cannot add an older receipt afterward. Neither participant's later policy requires the coordinator merely because onboarding used MuSig2.

10.3 Withdrawing donations without an identity action

```
Input: an output under the current TULIP policy
Authorization: an ordinary valid spend
Outputs: payment destination and change
Recognized identity action: none
```

Indexer: identity epoch and status unchanged.

The identity remains registered even when this consumes its last current output. Fresh funding can later supply another action opportunity.

10.4 Rotation or fresh-key recovery

```
Input: a UTXO under the current accepted (P_e, M_e)
Witness: recognized action path and its required authorization
Authenticated outputs/payload:
  ID, epoch e, predecessor, ROTATE or RECOVER,
  successor public key P_next and tree root M_next
  optional successor acceptance if required by this TULIP
```

RECOVER: publish the fresh public successor through OP_RETURN data.

Indexer: check target and unused key, advance epoch, reserve P_next.

ROTATE must match its committed succession rule. RECOVER uses the independently secured authority granted discretion to choose a fresh target. Neither universally requires a fresh signature from P_e. Any required successor output must match the authorized policy; exact output requirements remain to be specified per profile.

10.5 Recovery versus revocation in one block

Confirmed sequence	Result
Valid recovery at transaction 4; old-epoch revocation at transaction 7	Recovery wins; the stale revocation has no identity effect.
Valid revocation at transaction 4; recovery at transaction 7	Identity is terminally revoked; recovery has no effect.
Both actions in one transaction	First valid action by canonical action position wins.

These outcomes assume the stated actions target the same current epoch. An earlier unknown or invalid record does not defeat a later valid action.

10.6 WoT grouped update

Input/witness: current-policy UTXO and authorized WoT reveal
Authenticated OP_RETURN data, conceptual groups:
 weight 8, add/amend, count n: pubkey list
 weight 0, count m: pubkeys to remove
 weight 16, add/amend, count k: pubkey list

Each group has one shared weight and a proposed two-byte control header.
Indexer: apply WoT rules; identity epoch remains unchanged.

This does not allocate header bits or settle add-versus-amend encoding. Replacement of the entire set requires its own unambiguous operation boundary. A recipient does not sign merely to be listed.

11. Specification and implementation work remaining

The architectural decisions above are the current proposal. These items remain open before an interoperable protocol or security claim can be finalized.

Area	Remaining work
Shared public encoding	Canonical header and entry encoding; solo and MuSig2 authentication; provenance-mode binding; evidence availability; domain tags and versioning.
Lost-key public entry	Exact solo profile combining prior signed enrollment, committed action authority, public registration, and optional immediate transition without requiring the lost identity key.
Ordering and failures	Canonical within-transaction and within-batch positions; multiple carriers and actions; duplicate handling; whole-batch versus per-entry failure rules.
Permanent key registry	Exact reservation checks and timing, including precommitted but not yet accepted future keys; undo/replay; current and historical lookup.
Policy scripts	Concrete rotation, recovery, revocation, and WoT templates; signature coverage; preimage and threshold rules; successor binding and any required outputs.
Cryptography	Pinned setup artifact and digest; scalar hash-to-field; domains, padding, count and validation rules; receipt format; independent test vectors.
WoT layer	Two-byte header layout and count bounds; default-list discriminator; add/amend/replace semantics; authorization and witness binding; bootstrap signer roles; mapping of private/unregistered targets.
Coordinator scope	Defined non-key delegation, expiry and update authority; receipt delivery; participant review; MuSig2 abort/retry and nonce lifecycle.
Data and fees	Complete serialized transactions and receipts; carrier and relay limits; funding and change; optional-proof overhead; indexing and light-client requirements.
Fee-funded recovery	Actual regtest transactions, package acceptance and conflict/replacement experiments, accounting, and explicit implementation versions.
Extensions	Honeypot slots, inscription and asset links, other-chain profiles, and Nostr integration as separate specifications.

Implementation should begin with canonical records and an independent verifier, then demonstrate solo public registration, policy actions, optional seed batch provenance, and MuSig2 batching with reproducible vectors. Cryptographic libraries and coordinator implementations must be selected and reviewed; this paper does not supply custom cryptographic primitives.

References

1. Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.
2. Bitcoin Improvement Proposal 341. Taproot: SegWit version 1 spending rules.
3. Bitcoin Improvement Proposal 340. Schnorr Signatures for secp256k1.
4. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Polynomial Commitments, 2010.
5. Ethereum Improvement Proposal 4844. Shard Blob Transactions. Relevant here for KZG encodings and verification conventions, not as a complete seed-batch specification.
6. Bitcoin Improvement Proposal 342. Validation of Taproot Scripts.
7. Bitcoin Improvement Proposal 141. Segregated Witness.
8. RFC 9162. Certificate Transparency Version 2.0. Relevant here for binary Merkle inclusion proofs.
9. Ethereum Foundation. KZG Ceremony Specifications.
10. Ethereum Foundation. Wrapping up the KZG Ceremony, January 23, 2024.
11. RFC 9380. Hashing to Elliptic Curves, including hash-to-field and domain separation.
12. Bitcoin Improvement Proposal 431. Topology Restrictions for Pinning.
13. Bitcoin Core. Mempool Replacement Policy. Relay policy can change independently of consensus.
14. Nostr Implementation Possibility 01. Basic protocol flow description.
15. Nostr Implementation Possibility 07. Window.nostr capability for web browsers.
16. Bitcoin Core. getblockchaininfo RPC documentation, including pruning status and retained block range.
17. Bitcoin Improvement Proposal 327. MuSig2 for BIP340-compatible Multi-Signatures.
18. Bitcoin Developer Reference. Transactions, transaction-output serialization and CompactSize encoding.
19. Bitcoin Core. Transaction and relay policy constants. Policy limits are implementation-specific and may change independently of consensus.
20. Bitcoin Improvement Proposal 112. CHECKSEQUENCEVERIFY, relative timelocks based on the age of the spent output.
21. Bitcoin Improvement Proposal 433. Pay to Anchor (P2A).
22. Bitcoin Core. Package Mempool Acceptance Rules.
23. Bitcoin Core. Ephemeral dust policy implementation.
24. Bitcoin Core. Dust threshold calculation.