

Tuliproot: A Bitcoin Metaprotocol for User-Controlled Identity

Michael Tidwell
michael@tidwell.io
tuliproot.com

Draft 0.6 | October 23, 2026

Abstract

A public key lets someone prove control of a secret. It does not explain what should happen when that secret is lost or stolen. Tuliproot lets users set those rules in advance. Identities can be seeded in large private batches, with one Bitcoin commitment recording the batch. A user can later publish their identity and its policy, together with proof of its earlier seeding, or start directly onchain. TULIPs combine identity keys with Taproot policies to control key rotation, recovery, and revocation. The identity survives key changes and ordinary spending. Users keep their own proofs and recovery material, so they can act without the original coordinator.

1. Introduction

An account needs more than a signing key. It needs a way to replace that key without losing its history or asking an account provider for permission. Giving the everyday key full control over its own replacement makes theft harder to recover from. Losing that key can leave an account unusable even when its owner still has other ways to prove authority.

Tuliproot separates everyday signing from the rules for changing an identity. It uses three ideas:

- **Seeding** uses the identity key (the Taproot internal key) to sign a record binding its policy tree (MAST) into a private batch. One onchain commitment records the batch; each user keeps a proof of their entry.
- **Blooming** publishes the key and policy on Bitcoin, either in a MuSig2 batch or alone. A user may include their seed proof or begin without one.
- A **TULIP**, or Taproot UTXO-Linked Identity Program, combines the identity key and its policy. Its Taproot address receives funds, and its spending paths control rotation, recovery, and revocation.

Seeding is optional. Acting onchain requires blooming and funding a UTXO at the TULIP address.

A **Tulip identity** is the persistent identity. Its **identity key** is the current Taproot internal public key. It keeps the same identity ID through accepted key changes. Applications decide what its history is worth.

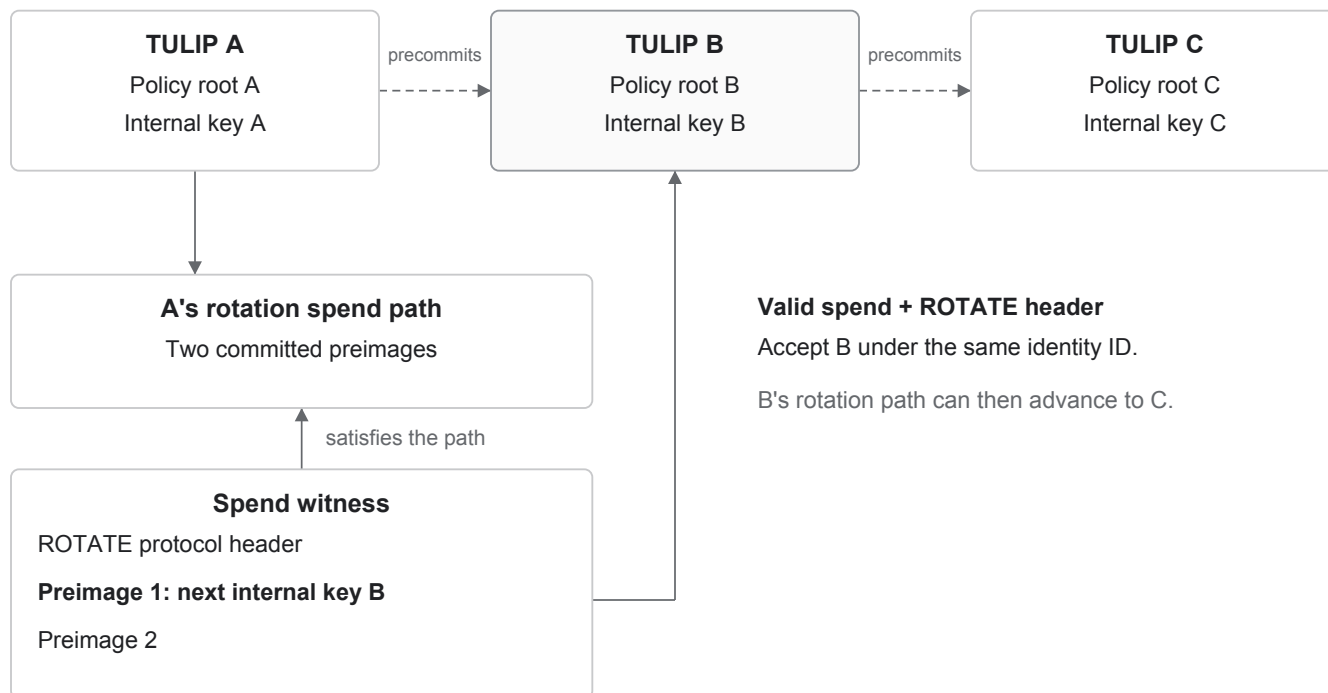
2. TULIPs and rotation chains

A TULIP combines an internal identity public key with a Merkle tree of spending paths (MAST). The key and tree root determine its Taproot address. Its paths set the conditions for spending a UTXO and performing authorized identity actions.

A rotation path commits to the next TULIP, which can commit to another. Each valid rotation changes the accepted key and policy while preserving the identity ID.

A valid rotation spend reveals two committed preimages and the proper ROTATE witness header. One preimage supplies the next TULIP's internal key. Any signing requirements come from that TULIP's policy.

Figure 1. A precommitted rotation chain



3. Seeding identities at scale

A user chooses an identity key and a Taproot policy. The policy contains the rules for future rotation, recovery, and revocation. The user signs a seed record that binds the key, policy, coordinator, and batch. The coordinator collects these records and publishes one commitment on Bitcoin.

Each user receives their signed record, a proof of inclusion, and a reference to the confirmed batch. Together, these show that the signed record existed by that block. The coordinator need not publish the individual records or remain available afterward.

For example, 30,000 Nostr users could each create a TULIP using their existing identity key and chosen policy, then join one private batch. The proposed KZG scheme over BLS12-381 records the batch with a single 48-byte commitment.[4][6] The coordinator pays one shared Bitcoin transaction fee, with batch metadata and transaction overhead; it need not create 30,000 outputs. Each participant keeps a 48-byte inclusion proof (64 base64 characters), their signed record, index, and batch reference. A proposed Nostr kind-0 profile field could carry that evidence.[11]

When blooming, the proof adds 48 bytes and a batch index can use 2 bytes. A 32-byte seed-transaction ID locates the shared batch declaration, giving an 82-byte proof-and-reference subtotal. The original 64-byte enrollment signature brings the fixed provenance components to 146 bytes, plus any signed fields not already available from the public entry or batch declaration. Entrants from the same batch can share the transaction reference. These figures exclude the public identity entry and transaction overhead; they do not require republishing the seed transaction.

4. Blooming onchain

Blooming publishes the identity key and policy root and establishes the permanent identity ID.[2]

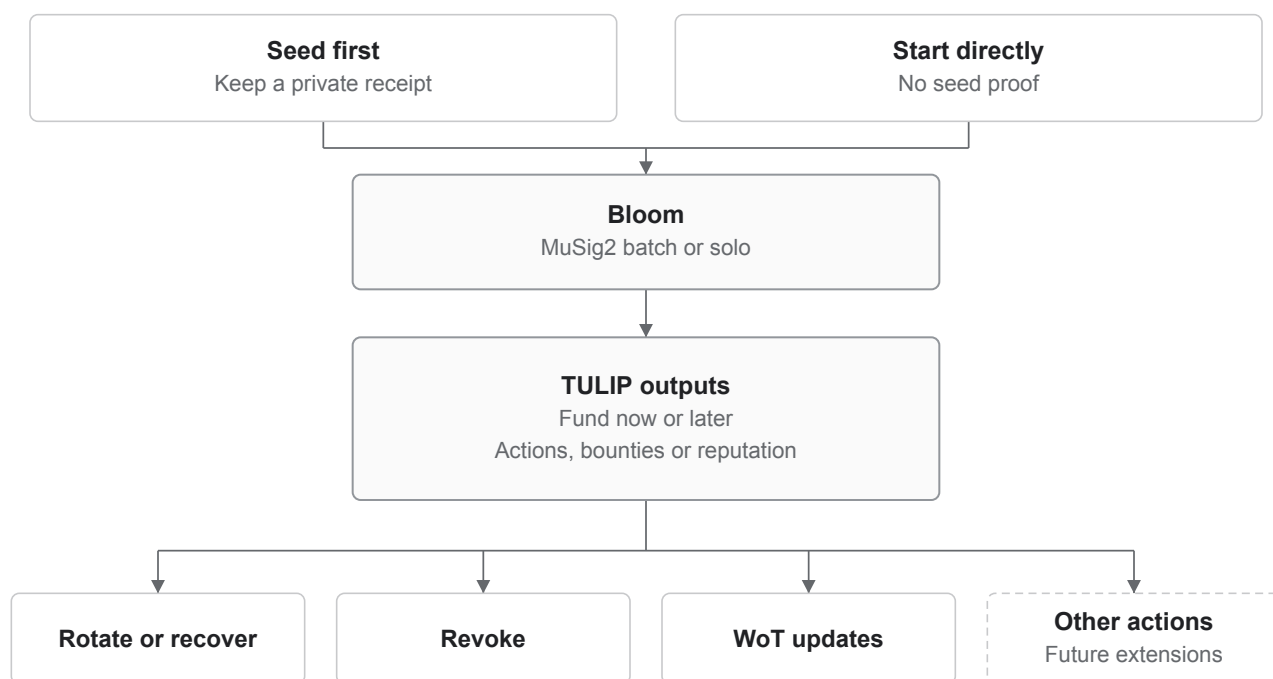
A coordinator can collect many entries into one manifest and use MuSig2 to obtain their joint consent.[5] Each participant approves their own entry and the shared message. The single aggregate signature authenticates the batch; it gives neither the coordinator nor the other participants control over an individual's TULIP. A user can perform the same public step alone.

A seeded participant includes their signed record, opening, and batch reference with this first public entry. The proof establishes earlier history. A participant without that evidence starts fresh. Proofs cannot be added later to upgrade the identity's age.

The transaction may create each participant's TULIP output immediately, or leave funding until later. In either case, the public identity follows the same rules.

Funding can also precede any planned action. A public bounty or honeypot can put visible value behind an identity's reputation.

Figure 2. From optional seeding to TULIP actions



5. Using a TULIP

Each payment to a TULIP address creates a separate unspent transaction output, or UTXO. Any suitable output under the current TULIP can be spent to perform an authorized action.

Action	Purpose
Rotate	Move to a successor committed by the rotation policy.
Recover	Use recovery authority to select a fresh key and policy.
Revoke	End the identity permanently.
WoT update	Publish or change outgoing vouches; described in Section 9.

Ordinary spending leaves the identity unchanged. Someone who receives donations at a public TULIP can withdraw them without rotating, recovering, or revoking. Even spending the last output does not end the identity. Another output can be funded later.

A key change advances the identity to a new TULIP. Remaining funds at the old address are still spendable, but later spends from that old policy cannot change the identity.

6. Rotation and recovery

Rotation follows a succession plan committed in the policy. Recovery allows its designated authority to select a fresh key, published in the action's OP_RETURN data. Both preserve the identity ID and can work after the everyday identity key is lost.

The owner chooses the conditions. A separate offline key, a k-of-n threshold, or a guardian arrangement can authorize an action. Shamir secret sharing can back up a separate authorization key. Timelocks can provide additional options. These are policy choices rather than one mandatory recovery scheme.

The recommended setup uses separate rotation and recovery keys, independent of the everyday identity key. A signature from the new identity key can confirm that its secret is available before a transition is accepted. This is also recommended, rather than required for every TULIP. It does not require access to the old lost key.

For example, Alice loses her everyday key. Her committed recovery authority authorizes a fresh key and policy. Once that action confirms, services accept the new key under Alice's original identity ID. The lost key no longer controls the identity's next action.

Assume the entire script tree is public. Hidden branches and private transaction submission are not security foundations. A policy that lets the everyday key alone change the identity remains vulnerable to theft of that key. Compromise of the recovery authority itself should be treated as compromise of the account.

7. When an attacker gets a TULIP's internal key

An exposed internal key lets an attacker spend TULIP funds through the key path. Rotation or recovery can still work if its separate spending conditions remain secure. The attacker may try to block that action by spending the same output first.

A proposed defense separates the exposed value from the fee budget:

1. A zero-fee parent spends a confirmed input under an uncompromised key, creating a small output to a TULIP address (330 sats in this example).
2. A child spends that output through the rotation or recovery path. A separate, confirmed input under an uncompromised key pays the package fee. If this child loses the conflict, that fee input remains unspent.

Both transactions use version 3 (TRUC), which limits the unconfirmed package to a parent and child.[7] An optional pay-to-anchor (P2A) output can be placed in the parent and consumed by the same child. A zero-value anchor requires the applicable ephemeral-dust rules; it does not add another unconfirmed transaction.[9]

Replace-by-fee (RBF) lets a higher-paying conflict replace an unconfirmed transaction. At nodes enforcing the referenced replacement policy, the attacker's smaller key-path spend must cover the evicted fees, the

replacement relay charge, and the other fee requirements. A cheaper spending path does not bypass the absolute fee bid.[8]

Suppose the owner offers 20,000 sats to recover an identity whose internal key is fully exposed. If the recovery child is accepted and the attacker replaces it, the conflicting spend must meet those replacement requirements. If the attack confirms instead, the owner loses the 330-sat output; the protected 20,000-sat fee input is not spent. The attacker pays its confirmed transaction fee, offset by any stolen value. A fee above 20,000 sats therefore costs the attacker more than 19,670 sats of its own funds after capturing all 330 sats.

The owner can fund another attempt, making each confirmed blocking spend cost the attacker again. Unconfirmed replacement bids alone pay no fees. If recovery succeeds, the owner pays its actual fee. Private submission may reduce exposure to a race; public submission allows fee competition. Neither guarantees confirmation or forces miners to honor mempool policy.

The 330-sat figure is the usual Taproot dust threshold under default policy. The 240-sat figure belongs to P2A, not a TULIP output.[9][10]

8. One accepted history

The first valid public registration claims the identity key. Later registrations using that key are ignored, including ones with different Taproot trees. An earlier private seed receipt proves age but does not reserve the key. Once a key has been accepted for an identity, it cannot be reused, even after rotation, recovery, or revocation.

Competing actions follow confirmed chain order. Within a block, the earlier transaction wins. Within a transaction, the first valid action in the defined action order wins. Invalid actions do not change state. If recovery wins first, a later revocation under the old policy is ignored. If revocation wins first, the identity is dead.

Indexers apply these rules to find the current key. A chain reorganization requires them to undo and replay the affected history.

9. Weighted web of trust

Weighted web of trust (WoT) is the first extension proposed alongside Tuliproot: a metaprotocol built on its identity history. An identity publicly vouches for another, using weights to express a lesser or stronger relationship. The recipient need not approve or return the vouch. Applications decide how to interpret the relationships and weights.

WoT indexers can follow a target through rotation and recovery; revocation ends the live connection. A vouch may name an unbloomed key, although publishing the vouch reveals that key. The technical draft defines the proposed actions, weight range, defaults, and grouped updates.

10. Cost, privacy, and verification

KZG keeps inclusion openings small but introduces pairing cryptography and a trusted setup. A conventional Merkle profile would trade larger proofs for simpler assumptions.

Blooming has a different cost. MuSig2 combines consent into one 64-byte signature, while the public keys, policy roots, and optional seed evidence still take space.[3][5] Creating TULIP outputs and making later changes adds transaction costs. Neither form of batching makes an identity's full lifetime free.

Seeding can leave individual records private. Blooming reveals the identity key and policy root, and attaching a seed proof links their earlier history. A script-path spend reveals the branch used. Users should choose disclosures with these links in mind.

Services can verify the public history themselves or rely on an indexer. A receipt proves earlier seeding, not the current key. An indexer must also account for accepted key changes and revocation. Bitcoin validates the spends; identity software checks the protocol records.

An old identity, a funded address, or an untouched honeypot does not prove good behavior or that a secret has never leaked. The system provides evidence, not a universal reputation score. Nostr can use the same Schnorr keys, while clients would need additional rules to follow identity changes.

11. Conclusion

Tuliproot gives users a way to establish history in batches and change keys under rules chosen in advance. Seeding keeps initial records private. Blooming publishes the identity when public management is needed. TULIPs provide rotation, recovery, and revocation without making the identity depend on one coin or one coordinator. Higher layers such as WoT can follow those changes while making their own judgments about trust.

Implementation notes

This is a proposal, not a finished wire protocol or an audited implementation. The technical specification draft retains the cryptographic construction, verifier rules, byte accounting, transaction examples, and remaining implementation questions. The archived detailed draft 0.5 preserves the previous paper in full.

The implementation guide covers coordinators, solo operation, verifier storage, and testing.

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. Bitcoin Improvement Proposal 327. MuSig2 for BIP340-compatible Multi-Signatures.
6. Ethereum Improvement Proposal 4844. Shard Blob Transactions, for the referenced KZG encodings.
7. Bitcoin Improvement Proposal 431. Topology Restrictions for Pinning, version-3 transaction topology.
8. Bitcoin Core. Mempool Replacement Policy.
9. Bitcoin Improvement Proposal 433. Pay to Anchor (P2A), including its distinction from ephemeral dust.
10. Bitcoin Core. Dust threshold calculation.
11. Nostr NIP-01. Basic protocol and kind-0 metadata.