t1 Vision Litepaper

Last modified: November 10, 2025

https://t1protocol.com

Introduction

Abstract

t1 is the first interop rollup designed to fix fragmentation and composability challenges in scaling Ethereum. By leveraging AVS-secured Trusted Execution Environments (TEE), t1 introduces Real-Time Proofs (RTP) that prove the integrity of t1 execution to Ethereum in less time than it takes to create a block on Ethereum (12 seconds). By running third-party *Partner Rollup* nodes in its TEE node infrastructure, t1 also aggregates and proves their needed state to Ethereum in real-time. As a result, t1 enables instant settlement between any combination of *Partner Rollups* and Ethereum L1, providing fast composability.

Execution in t1 is cryptographically verified via zk-compressed TEE Remote Attestation, ensuring that all involved state transitions are provably correct, tamper-proof, and bound to the current protocol version. In addition to real-time proving, t1 supports general-purpose smart contract programmability in its blockspace, and enables writes to *Partner Rollups*. This architecture offers a foundation for building new and enhancing existing cross-chain applications—such as yield aggregators, lending protocols, and decentralized exchanges—that require fast, programmable interoperability without reliance on third-party bridges or message-passing systems.

t1's mission is to unify Ethereum and its rollup ecosystem by creating a real-time proof-powered liquidity layer that enables the best user experience on cross-chain applications.

Table of contents

Problem

The current rollup-centric scaling strategy has delivered lower fees and higher throughput, at the cost of fragmentation. Liquidity and user activity are now spread across hundreds of rollups, forcing developers to bring their applications multi-chain—by leveraging third-party bridges and messaging protocols. Despite widespread efforts to improve interoperability, the current landscape remains siloed:

- **Most L2 interoperability efforts focus on specific rollup stacks** rather than the entire ecosystem. These efforts reduce the number of siloes yet solidify existing siloes rather than eliminating them.

- **Interoperability protocols enable message passing and proof generation, but forego the benefits of programmability.** As a result**, each chain remains a siloed execution environment.** Users must take separate actions on each chain, adding friction, cost, and fragmentation.
- **Real-Time Proving in zero-knowledge**, especially for L2-level compute throughput, is still far out, both because of proving times, but also due to the complexity and bug potentials in zkVMs.

As a result, no existing application can offer a seamless, cross-chain user experience today without the complexities and risks of introducing third-party bridges or messaging protocols. Without a fundamental shift in how interoperability is achieved, Ethereum risks developing into a fragmented network where composability remains trapped within silos.

Solution

t1 introduces a Trusted Execution Environment (TEE) Real-Time Proving (RTP) rollup to solve fragmentation in the Ethereum ecosystem today.

Real-Time Proving

RTP allows t1 to prove its state to Ethereum L1 on every L1 block. This means:

- Users can enjoy the low cost and high speed of rollups while their assets may effectively be viewed as remaining on Ethereum: Funds on t1 can be withdrawn to an L1 account within a single block, making interacting with t1 feel similar to interacting with a smart contract on L1.
- Applications on Ethereum and on *Partner Rollups* can quickly access t1 state (incl. liquidity) because it can be proven to Ethereum on every L1 block.
- Applications on t1 can verify state changes in Ethereum and *Partner Rollups* and create cross-chain proofs. This is achieved by running *Partner Rollup* follower nodes within t1.

Programmability

Programmability enables t1 to do more than just Real-Time Proving. Smart contracts create a programmable hub for liquidity and cross-chain interactions. Programmability enables components currently deployed off-chain, such as relayer and solver networks for ERC-7683 protocols, to be brought on-chain. Programmability also enables t1 to become a cross-chain liquidity hub in addition to just being a bridge or a proving system that merely passes proofs across blockchains. This means we can now build collateral accounts, lending-borrowing primitives, and orderbooks—but in the cross-chain paradigm.

Furthermore, t1's architecture is designed to allow for cross-chain application experiences without needing buy-in from any rollup/L2 or application. In that sense, t1 is the much-needed permissionless cross-chain application platform.

What are TEEs?

Trusted Execution Environments (TEEs) are specialized hardware-based environments that isolate sensitive computations and data from the rest of the system, ensuring that data is processed correctly and (optionally) privately.

In particular, TEEs provide verifiable computation guarantees through a process called "Remote Attestation" which proves to external verifying parties that the TEE in question is running a specific, unmodified piece of software (bytecode), without any tampering. Verifiers can then use this attestation and combine it with an understanding of what the bytecode is doing in order to confirm that a TEE's output is indeed trustworthy.

Optionally, TEEs can preserve privacy by keeping sensitive data and execution logic concealed from the broader system and external observers.

How do they help?

Two key requirements for achieving full interop unification of Ethereum and the rollup ecosystem, without reorg risks and asynchrony, would be shared sequencing across all chains, and real-time proving (RTP). The former does not seem realistic at this point.

At t1, we are working on RTP by employing TEEs. However, TEEs also help with cross-chain composability by enabling follower nodes in t1 to read data from and write data to *Partner Rollups*. This setup allows t1 to aggregate the state of Ethereum and *Partner Rollups* without requiring shared sequencing (or any opt-in), with a best-effort approach where the asynchrony is not bounded in theory, but low in practice.

And again, RTP enables t1 state (which is dependent on *Partner Rollups'* state implicitly) to have an asynchrony window with Ethereum that is as low as a single-block (12 seconds)—a substantial improvement over the current seven-day window in Optimistic Rollups and hours-long window in Zero-Knowledge Rollups.

In addition to RTP and cross-chain communication, TEEs allow t1 to offer a partially-encrypted mempool. Such a mempool prevents adversarial reordering, such as sandwich attacks, where an attacker observes a pending user transaction and places trades before (front-running) and after (back-running) it, profiting at the expense of regular users. Sandwich attacks cost Ethereum users over $100mn every year [1]. An encrypted mempool or ephemerally-private blockspace may also facilitate use cases like sealed-bid auctions and information-incomplete games.

Architecture

Protocol design

t1 is an EVM-based rollup that generates real-time proofs to provide cross-chain application infrastructure. In its currently planned long-term form, t1 combines the verifiable computation guarantees of Trusted Execution Environments (TEE) with additional defense layers such as economic security (AVS) and bespoke zero-knowledge proofs (ZKP) to enable fast and secure proof generation. t1 has two network stakeholders:

- *Sequencers* are a highly decentralized set of nodes tasked with blindly finalizing the ordering of partially-encrypted transactions in a t1 block. Since Sequencers only order transactions rather than executing them (meaning lower hardware and network requirements, in particular no TEE requirement), t1 can achieve high decentralization and censorship resistance. Sequencers provide proofs of *Sequencing Consensus*. More: .
- *Executors* are a network of TEE-enabled nodes tasked with executing state changes given the ordered sequences of transaction bundles determined by the Sequencers. Executors provide proofs of *Execution Consensus*. More: .

t1 produces blocks every second. Each block involves two sequential steps:

- Tx data broadcast and bundle finalization by *Sequencers* (incl. *Sequencing Consensus* proof).
- Execution into a block and agreement about the new state, reached by TEE-enabled *Executors* (incl. *Execution Consensus* proof).

Both *Sequencing Consensus* and *Execution Consensus* are required to update the state of t1 (aka trie root tuple) on Ethereum. Only then will t1 bridge contracts on Ethereum and *Partner Rollups* be able to be “convinced” to, e.g., release their funds to withdrawing users.

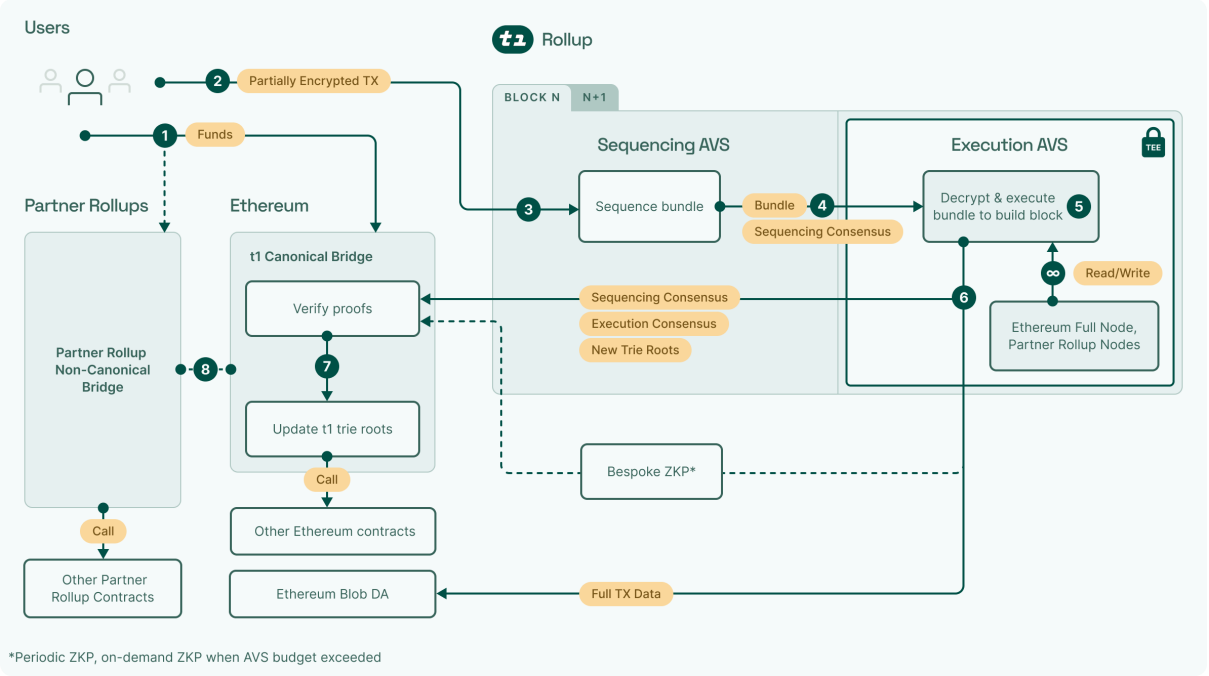
As t1’s interim form gradually becomes a fully permissionless network, it’s essential to implement mitigations against potential TEE exploits and develop defense-in-depth strategies. To this end, t1 leverages two sets of EigenLayer Autonomous Verifiable Services (AVS) validators to derive its crypto-economic security from restaked assets, providing a programmatic insurance budget on top of TEE guarantees.

However, an attacker controlling more than the crypto-economic security of the Execution AVS stake and also the necessary TEEs (that they managed to compromise) could produce an integrity proof for a fraudulent new state of t1. For this attack to be economically viable, the value-at-risk would need to be higher than the slashable Execution AVS stake. To ensure that t1’s economic activity is not bound by this security budget, we introduce a *bespoke ZKP*

mechanism as an additional defense layer: t1 uses incentivized *periodic ZKP* to create *checkpoints*. When cumulative value-at-risk since the last *checkpoint* is about to exceed the crypto-economic security budget, the *Canonical Bridge* will require the provision of an *on-demand ZKP* before it accepts such a new t1 state—halting finalization until then. However, with adequate t1 gas price policies incentivizing proactive *checkpoint* creation before any finalization halting is needed, we don't expect this situation to ever happen under normal conditions, achieving a good tradeoff between finalization latency and hard-times resilience.

Note: ZKP generation will likely be outsourced to a network that can meet the latency and cost requirements.

Architecture diagram



Flow

1. A user, Alice, deposits funds to a t1 bridge contract on Ethereum or on a *Partner Rollup*. Once the deposit is confirmed on the source chain, it gets processed by t1, and Alice gets her funds credited towards her aggregate t1 balance.
    - Note: Recent progress on Fast Confirmation Rule (FCR) (aka *Fast Synchronous Finality / FSF*) directly contributes to t1's ability to interpret both Ethereum's state and also "secure" *Partner Rollups*' states as *sufficiently finalized* significantly faster than under the standard *finalized*-tag (~16 minutes) assumption—drastically lessening the risk of t1 needing to reorg together with an L1 reorg (or only being able to credit user deposit/inbox transactions with a *finalized*-tag-level delay).

- This is true for a *Partner Rollup* if it is parameterized to operate in the most secure mode of t1 only deeming such *Partner Rollup* states as final whose DA has been posted to Ethereum **and** is viewed as *finalized* there. Less secure and significantly faster modes for *Partner Rollups* are possible which introduce the trust assumption in the *Partner Rollup* sequencer and depend on its p2p/gossip network layer. Hybrid modes with capped value-in-flight are conceivable, too.

2. Alice changes her wallet's network to t1, creates a t1-native transaction (with some fields encrypted to the shared rotating TEE pubkey), uses her wallet to sign it, and submits it to the network (i.e. the t1 mempool); this may or may not be a specially-treated withdrawal transaction (to Ethereum or a *Partner Rollup*).
3. A t1 *Sequencer* receives and gossips such a partially-blind transaction to other *Sequencers* in the t1 *Sequencing AVS* network.
4. After collecting transactions for one t1 slot (currently set to one second), the slot-leading *Sequencer* proposes an ordering (a blind non-executed bundle). The rest of the *Sequencers* vote on it using Espresso HotShot to form *Sequencing Consensus*. This bundle and a proof of *Sequencing Consensus* is then passed on to the *Execution AVS* network.
5. t1 *Executors* validate the proof of *Sequencing Consensus*, decrypt the encrypted parts of the received bundle (if needed and due) using their TEE-derived shared rotating private key, and execute its now fully plaintext ordered transactions against the current state of the t1 blockchain. The slot-leading *Executor* proposes a new trie root tuple *r* of state trie root, withdrawals trie root, and proof-of-read trie root—and the rest of the *Executors* vote on such new trie tuple *r* to form *Execution Consensus*.
    - Note: *Executors* use follower nodes also running in TEEs to read from and write to *Partner Rollups* (whenever required by a t1 tx).
6. The *Execution* AVS posts t1's new trie roots *r* and all the corresponding consensus proofs to the Ethereum t1 *Canonical Bridge* contract and the full compressed transactions to Ethereum blob DA.
    - In addition, t1 progressively incentivizes the generation and posting of *periodic ZKPs* to the *Canonical Bridge* on Ethereum to create *ZKP checkpoints*, resetting the value-at-risk counters and also speeding up the potential *on-demand ZKP* creation when required. t1 dynamic gas pricing considers how much AVS security budget is still available to reach an equilibrium.
    - In the rare event that new t1 transactions' (as per all new trie root tuples) cumulative value since the last *ZKP checkpoint*, despite the mechanisms above, would exceed

the crypto-economic security budget provided by *Execution AVS*, also an *on-demand ZKP* is required by the *Canonical Bridge*, pausing finalization until then; this would increase the withdrawal delay to hours under such extreme conditions.

7. t1's *Canonical Bridge* contract on Ethereum checks the new submitted t1 trie root tuple *r*, *Sequencing Consensus*, *Execution Consensus*, and transaction data availability for consistency. If successful, such *r* is accepted. This then generally facilitates withdrawals from t1 to Ethereum with a single-Ethereum-block delay only (i.e. 6 seconds on average).
    - Suppose Alice had desired to withdraw funds in step 2. She may now submit to the *Canonical Bridge* an Ethereum claim transaction with an inclusion proof of her withdrawal transaction in t1 (as contained within the withdrawal trie committed to in *r*). The contract then releases the funds to Alice on Ethereum.
8. If Alice wishes to withdraw funds to her account on a *Partner Rollup* rather than on Ethereum, the same trie root tuple *r* update in the *Canonical Bridge* (i.e., on Ethereum) is required as in 7. However, she submits the claim transaction with an inclusion proof of the withdrawal to the (non-canonical) t1 bridge contract on *Partner Rollup* instead. The *Partner Rollup* bridge contract verifies the inclusion proof with respect to *r* as accepted by the *Canonical Bridge* on Ethereum (using *Partner Rollup*'s Ethereum read abilities, usually via *Partner Rollup*'s own L1 canonical bridge) and then releases the funds to Alice on *Partner Rollup*.
    - Note: The same Fast Confirmation Rule as above lets *Partner Rollup* sequencers read the new *r* from the *Canonical Bridge* significantly faster.

Bridge contracts

t1 leverages its contracts on Ethereum and rollups to accept deposits into t1 and allow users to withdraw funds from t1 back to their blockchain of choice.

Ethereum's contract acts as the *Canonical Bridge* where t1 trie root updates are posted. Their validity is ensured by verifying *Sequencing Consensus*, *Execution Consensus*, and the data availability commitment to a blob posted on Ethereum (DA). Therefore, **this contract is the source of truth for t1 state.**

Withdrawals to Ethereum can be enabled immediately after a transaction updates this contract, by a further "claim" transaction carrying a Merkle proof, on L1 or *Partner Rollup* itself. Withdrawals to *Partner Rollups* require the local (non-canonical) t1 bridge contract to check the state of the *Canonical Bridge* on Ethereum before the funds can be released to the user on *Partner Rollup*. The state of the t1 *Canonical Bridge* can be relayed to *Partner Rollup*

via arbitrary message passing capabilities of *Partner Rollup*'s own bridge. This approach prevents attacks that could otherwise result in double-spend due to reorgs.

Sequencers

In t1, Sequencers sequence and broadcast partially-encrypted transactions, forming a consensus on a blind block order ("bundles"), which is then passed to Executors for execution. Most of the rollups nowadays utilize single-sequencer designs, compromising security for speed; also, usually both sequencing and execution are performed by a single entity, leading to little censorship resistance. We want to separate sequencing (broadly accessible) and execution (higher network and hardware requirements, including the requirement of being a TEE) to allow for fast, decentralized consensus, offering strong resistance to censorship and bribery, while minimizing MEV from transaction ordering.

t1's long-term design employs Espresso Systems' HotShot Consensus, a BFT protocol adapted for PoS, enhancing decentralization. Sequencers, part of t1's AVS, use restaked stake with slashing conditions to maintain security and integrity in transaction sequencing.

Executors

TEE-enabled Executors oversee the current state of t1 by running transactions from Sequencer-ordered blind bundles and establishing consensus on the new state. They utilize t1VM, a TEE-optimized and interop-enriched version of Reth, which ensures full backwards compatibility with Ethereum while adding cross-chain-composability-specific capabilities such as reading a chain's contract method or requesting a write call to it. An Ethereum full node is maintained within the Executor for streamlined interaction with the canonical bridge, while follower nodes (clients) for other rollups run by the Executor allow for real-time state monitoring and interactions with them, without the delays usually required for fraud or zero-knowledge proofs. t1VM includes unique capabilities powered by Reth's ExEx, enabling Executors to interact with both Ethereum and other (partner) rollups from within t1's Solidity smart contracts.

Executors operate under a leader-based, instant-finality PoS Byzantine Fault Tolerant consensus system, representing the second category of t1's AVS, and they are liable to slashing to enforce system integrity.

All execution happens inside Intel TDX-based TEEs, which cryptographically isolate the runtime logic and protect it from tampering, even by the host OS. Before processing transactions or joining Execution Consensus, each Executor must successfully complete a hardware-backed Remote Attestation, as detailed in the Remote Attestation Design and Enforcement.

Special opcodes for cross-chain composability

The *t1VM*, based on Reth and optimized to run within Intel TDX enclaves, introduces several custom precompiles and execution extensions (ExEx) to support seamless cross-chain programmability. These opcodes are enclave-executed thanks to co-located Partner Rollup follower nodes.

These special instructions allow t1 smart contracts to:

- `xchain.readState(uint64 chain, bytes calldata data)`

  Query the state of a remote Ethereum or Partner Rollup contract in near-time, using enclave-hosted follower nodes. Reads are cached and committed via proof-of-read trees in block headers.

- `xchain.sendTx(uint64 chain, bytes calldata data)`

  Submit transactions to Ethereum or Partner Rollup mempools from inside a t1VM contract, enabling best-effort multi-chain coordination.

  While the transaction submission is initiated synchronously from within t1VM, its inclusion and execution on the target chain are asynchronous processes. Smart contracts must be designed to handle potential failure, delayed inclusion, or rollback by relying on callback patterns, event monitoring, and appropriate timeout handling.

- `xchain.sendTxAs(address user, uint64 chain, bytes calldata data)`

  Relay transactions to other chains *on behalf of a user*, enabling use cases like delegated execution, paymasters, or cross-chain proxy control.

- `xfeed.queryPrice(string feedId, uint64 timestamp)`

  Access a signed price feed snapshot (e.g., from a partner CEX or L1 oracle) as of a specific timestamp, as observed by the enclave. This enforces deterministic read behavior, allowing values to be committed into the proof-of-read trie and enabling reproducibility across replays. Suitable for use in collateral valuation, liquidation triggers, or TWAP rebalancing.

All read-based opcodes, such as xchain.readState and xfeed.queryPrice, are executed inside the TEE, and their outcomes are committed into a proof-of-read trie, which is included in the block header and committed to Ethereum. Remote reads are deterministically committed to Ethereum L1 via a proof-of-read trie, ensuring verifiability and auditability of all external dependencies in t1 state transitions.

In contrast, transaction submission opcodes, such as xchain.sendTx and xchain.sendTxAs, initiate asynchronous best-effort actions toward external chains. These are not committed to the proof-of-read trie. Instead, applications using these opcodes must be designed to handle

external success or failure events explicitly, through callback patterns, confirmation monitoring, or timeout strategies.

Further rollup node subnetworks

Additional subnetworks may appear in the broader t1 ecosystem where some TEE-enabled *Executor*-style nodes, probably running as an AVS, provide light-client-like “oracle” services to e.g. t1 *Executors* who wish to act upon the latest (low-latency) state in some other (possibly arcane/insecure) rollup/L2. These could come with an insurance fund or marketplace for reorgs and other rollbacks, e.g., caused by a technical issue, security council breach, etc.

Such a subnetwork may or may not be operated by the same physical node operator and on the same hardware as a t1 *Executor*. t1 is being built in a modular way such as to facilitate tangential applications of this sort via a robust shared infrastructure layer. It is conceivable that different smaller subnetworks may emerge to account for the different, possibly fine-granular security, latency, liveness, etc., requirements per Partner Rollup.

Mempool encryption

t1 has an encrypted mempool that is designed to eliminate ordering-related MEV. The system uses a rotating private key shared among t1 *Executors* that remains sealed within the TEE. As a result, transactions partially encrypted to the corresponding shared public key, which *Sequencers* order into partially encrypted bundles, can only be decrypted inside *Executors*, for execution, allowing the creation of the block with its new state. Once transactions are executed, their full plaintext content is made public, and anyone is able to reproduce the computed state.

Remote Attestation design and enforcement

A complete formal specification of the attestation pipeline can be found in this companion document: https://docs.t1protocol.com/concepts/resources#tee-architecture

t1 leverages Intel TDX-based Trusted Execution Environments (TEEs) at first, to ensure that state transitions are executed within secure, hardware-isolated environments. However, enclave security alone is insufficient without a mechanism to prove—publicly and succinctly—that these nodes are indeed running unmodified, audited software.

To that end, t1 runs the full DCAP quote-verification logic *inside* a Risc0 zkVM and compresses the trace into a Groth16 proof ≈ 200 bytes. A single proof is emitted only when a validator (i) first joins, (ii) upgrades to a new image hash, or (iii) refreshes its TCB after Intel raises the minimum SVN. `NodeRegistry` verifies that proof once, stores the validator’s pubkey

+ measurement as *active*, and thereafter needs only an ordinary ECDSA signature on each block.

Binding attestation to protocol state and mempool key

Each attestation report includes multiple fields:

- **`report_data`** — A 64-byte custom field set by the enclave at launch, it commits to
  - `validatorPubKey` – the enclave's long-lived signing key;
  - `currentProtoVersion` – governance-set tag bumped on every allowed-measurement change;
  - `HashPubHPKE` – the hash of the epoch mempool-encryption public key;
  - `nonce` – 256-bit challenge issued by `NodeRegistry` to prevent replay.
- **`RTMR3`** — A cryptographic measurement (SHA-384) of the enclave's runtime configuration, such as the Docker Compose manifest, root filesystem hash, and other application-specific initialization data.
- **`imageId`** — A unique identifier for the expected software binary or container image (e.g., a hash of the enclave codebase).

The zkVM verifies that the Quote's signature chain is intact, the `TCB SVN` ≥ `minTCB`, `REPORT_DATA` matches the supplied nonce and protocol parameters, and that the TD measurement appears in the on-chain allow-list Merkle root. The proof's public outputs (`validatorPubKey, measurementHash, HashPubHPKE`) are recorded in `NodeRegistry`; any later block signed by a key absent from this *active set* is rejected.

RA proof compression with ZKP

Once the TD Quote is generated by the Executor enclave, it is passed into a dedicated **zero-knowledge virtual machine (zkVM)**, such as **RISC Zero**, which serves as an off-chain attestation verifier. The zkVM performs a full cryptographic validation of the TDX quote, including all relevant fields, and outputs a succinct proof that can be efficiently verified on-chain.

The zkVM verifier performs the following steps:

1. **Quote Parsing**

   It deserializes the TD Quote structure and extracts key fields, including:

   - `report_data` – the verifier-defined 64-byte binding value,
   - `RTMR3` – the measured runtime hash,

- `imageId` – the identity of the runtime image,
- TDX report metadata and signature.

2. **Signature and Certificate Validation**

   It validates the **Intel DCAP certificate chain**, confirming the quote was generated by a genuine Intel TDX-capable CPU using a valid Provisioning Certification Key (PCK) signed by Intel. This ensures the enclave is rooted in hardware trust.

3. **Attestation Field Checking**

   The zkVM enforces that:

   - `report_data` matches the expected constants
   - `RTMR3` equals the known good runtime measurement corresponding to the declared `imageId`,
   - The claimed `imageId` is part of the protocol's published set of approved executor images.

4. **Public Output Commitment**

   After verifying the quote and all integrity constraints, the zkVM outputs the attested values— `imageId`, `RTMR3`, `report_data`, and an *attestation passed* flag—as public data in the proof's journal. These values are included as public inputs to the zkSNARK, allowing any on-chain verifier contract to enforce attestation rules based on them.

5. **Proof Generation**

   The zkVM emits a succinct zero-knowledge proof (e.g., a STARK or SNARK) that confirms:

   "A valid TDX quote signed by Intel proves the enclave with `imageId` ran a trusted runtime matching `RTMR3`, and correctly reported `report_data` = X."

   This proof is submitted alongside:

   - Node registration requests, i.e., in *NodeRegistry*,
   - Claims of protocol version compliance, e.g, in smart contract upgrades.

ℹ️ On-chain verifier contracts, such as the *Canonical Bridge* or *Node Registry*, do not inspect the raw quote but instead validate the zkSNARK. These contracts check that:

- The zk-proof is cryptographically valid and corresponds to the expected zkVM circuit
- The attested `report_data` matches the expected constants
- The `RTMR3` is one of the currently whitelisted runtime hashes for the declared `imageId`
- The `imageId` is recognized and not revoked
- The node is not already slashed or removed from the active set

By relying on zk-compressed attestation, t1 achieves a **trust-minimized, gas-efficient enforcement** of enclave integrity, with all sensitive quote validation logic occurring off-chain in a reproducible zkVM program.

Enclave-keyed access and upgrade safety

To decrypt and process encrypted user transactions, Execution AVS nodes require access to a shared mempool decryption key. This key is **sealed to the enclave** and is **only provisioned to nodes** that have passed **remote attestation**, verified via zk-compressed TD Quotes.

The attestation proof must confirm that the enclave:

- Runs a permitted executor `imageId`,
- Measures the expected runtime configuration (`RTMR3`),
- Reports a fresh `nonce` bound into `report_data`,
- Declares `validatorPubKey` and `HashPubHPKE` that match the on-chain epoch constants,
- Reveals a CPU/TDX `SVN` not lower than `minTCB`,
- Embeds the current protocol tag `currentProtoVersion` in `report_data`.

Only then can the node receive the decryption key from existing peers in the network. Key transfer is performed through an in-enclave, mutually-attested ECDH handshake:

- Both TDs verify that the counterparty's Quote hash and `validatorPubKey` are marked as *active* in `NodeRegistry`.
- They derive a session key from ECDH entirely inside their respective enclaves.

- The live node re-wraps the sealed `root_secret` under that session key and transmits it; the newcomer unseals it and deterministically derives its HPKE keys.

  Unverified or inactive nodes cannot complete this handshake and therefore never obtain the decryption key.

Whenever the protocol is upgraded—such as a new Canonical Bridge version, runtime logic revision, or dependency patch—t1 Security Council simultaneously (i) bumps `currentProtoVersion`, (ii) publishes an updated Merkle root of `allowedMeasurements`, (iii) may raise `minTCB`, and (iv) schedules an epoch-wide HPKE key rotation by emitting `HashPubHPKE'`. These values define the next valid executor configuration.

Execution AVS validators enforce this policy by accepting only nodes that submit zk-compressed attestation proofs verifying these exact values. Executors must re-attest using the new configuration in order to regain access to the sealed mempool key and resume consensus participation. Until they do, the attested-ECDH channel refuses to deliver `root_secret`, instantly disabling their ability to decrypt new transactions or sign blocks. This mechanism ensures that outdated or forked nodes are cryptographically excluded from transaction execution and state transition, providing strong upgrade enforcement at both the enclave and consensus levels.

By binding sealed key access to a multi-factor enclave attestation—checked off-chain in a zkVM and enforced on-chain by verifier contracts—t1 achieves robust upgrade safety and execution integrity without relying on manual trust or off-chain coordination.

Data availability

In addition to the new trie roots and *Sequencing Consensus* and *Execution Consensus* proofs going to L1, the full (yet compressed) transactions (inputs to the state transition function) are posted on Ethereum as blobs by the *Executors,* and their availability is checked by the *Canonical Bridge* when validating a new proposed trie root tuple. This enables anyone to recreate the state of t1 (but for what happened since the last Ethereum block, so on average for a maximum of 12 seconds) from trusting Ethereum and Ethereum alone. This also supports forced transaction inclusion.

Forced tx inclusion (incl. exit)

If the t1 rollup is down, any user will be able to submit their “self-sequenced” t1 transactions (that may include, e.g., exiting from a DeFi position and then withdrawing to Ethereum) to the *Canonical Bridge* contract on Ethereum.

Infrastructure partners

Automata DCAP

Automata provides an open-source implementation of Intel's DCAP remote attestation framework, enabling trusted enclave verification without relying on Intel's centralized services. t1 leverages Automata's DCAP tools inside Executor nodes to generate and validate TD Quotes, forming the basis for zk-compressed Remote Attestation proofs. This ensures that only nodes running verified enclave software participate in Execution Consensus, anchoring hardware trust into Ethereum via zk verification. Automata's work allows t1 to decentralize TEE validation, maintaining security even as the network grows permissionless.

EigenLayer AVS

EigenLayer's Autonomous Verifiable Services (AVS) system allows new protocols to inherit Ethereum's economic security through re-staking. Stakers from Ethereum can re-stake their assets, like ETH, into an AVS, which secures another service, beyond Ethereum, like *Sequencer* and *Execution Consensus* for t1. This model gives new chains like t1 the ability to bootstrap security without building independent validator sets (leading to inefficiently locked-up capital). EigenLayer includes strict slashing mechanisms to align validator incentives across networks, ensuring that re-staked validators are punished for misbehavior, maintaining strong decentralized security.

Espresso Hotshot Consensus

Espresso's Hotshot consensus enables finalizing block contents with low latency by utilizing a leader-based protocol that minimizes coordination overhead. HotShot employs a highly decentralized set of nodes, where blocks are proposed by a leader and validated through multiple rounds of voting to ensure consensus. The protocol is designed for scalability and achieving finality within a second, even under high network load conditions, all while maintaining decentralization and security by involving many participating nodes in the consensus process.

RFQ bridges

As t1 targets to solve cross-chain composability and UX challenges, it must provide a world-class experience for users wanting to withdraw tokens from t1 to partner chains. The t1 bridge contract on the destination chain first facilitates such a withdrawal. However, there's no guarantee that the t1 contract has enough assets for the withdrawal. If it doesn't, t1 will tap into RFQ bridge protocols, quickly moving funds between the different t1 bridge contracts in

the background, all to ensure users can seamlessly withdraw tokens to their address on any destination chain.

Security: On Reorgs

t1 is building a real-time proving rollup that enables near-instant cross-chain settlement without waiting for long confirmation delays. This significantly enhances user experience and application composability across Ethereum and partner rollups. However, this design introduces finality risk — the chance that a transaction could be rendered void due to a reorg in one of the chains involved. To address this, t1 employs multiple tactics outlined below.

Firstly, t1 generally reorgs together with the L1 (but cf. the impact of FCR under “Flow” above). When this happens, the user experience on t1 is identical to the user experience on the reorged L1. Basically, the transaction never takes place. It’s also worth noting, reorgs on Ethereum are rare: Since the Merge to Proof-of-Stake, ~0.059% of Ethereum blocks have been reorged (almost always one-block deep, with only 4 instances of 2-block reorgs). Moreover, rollups that use centralized sequencers have even lower reorg risk.

The main practical reorg risk arises when the source chain (from which a deposit was made to t1) reorgs after relevant funds were already withdrawn out of t1 to the third chain. In this rare scenario, t1 would be out of funds. There are some approaches we are considering to alleviate this problem:

- Inclusion preconfirmations: An inclusion preconfirmation is an optimistic crypto-economic guarantee that the deposit transaction will be included in the source chain. Since inclusion pre-confirmations are independent of both ordering and execution success, they are rather cheap. An inclusion pre-confirmation would guarantee that even in the event of a reorg on the source chain (e.g. a late block on Ethereum), a t1-depositing transaction will eventually make it to the source chain. However, its success is not guaranteed as a malicious user could e.g. have their funds spent before the block is created for which the preconf had been given. This would result in the t1-depositing tx to be correctly included, albeit as a reverting tx.
- Execution-success preconfirmations: These optimistic crypto-economic guarantees also bind the sequencing entity to ensuring that a given tx succeeds (i.e., not revert), not just that it be included. They are expected to be more expensive, but would allow for fast deposits secured up to the crypto-economic threshold of the slashable stake of the sequencing entity.
- Insurance pools: t1 can introduce an insurance pool for fast deposit-and-withdrawals and keep track of which portion of all deposits is final and which is still prone to reorgs, thus

also capping the maximum loss. This is effectively the purpose that solvers serve today in intent-based bridging and cross-chain swaps. Such instant deposit-and-withdrawal actions can require a certain premium fee that would go to an insurance pool so that potential source-chain-reorg-related losses can be covered by the insurance pool. In the absence of preconfirmations, the size of instant deposit and withdrawals would be limited by the size of the insurance pool.

Reorgs are a risk that needs to be accounted for in any on-chain architecture; although they are rare, they can happen. t1's approach to reorgs is risk-based and strikes a balance between enabling a good user experience, providing the right functionality, and accounting for the worst-case scenario.

Native t1 applications

Applications built natively on t1 leverage real-time proving and are designed to provide a better user experience with seamless cross-chain coordination. We're still early in uncovering all of the possibilities real-time proving enables, but have identified a few primitives that are uniquely enabled by t1:

- Cross-chain vaults, non-custodial cross-chain yield optimization that automates yield discovery and rebalancing across rollups. Using t1's real-time proving (RTP) interoperability infrastructure, funds move between lending protocols and yield sources.
- Cross-chain loans enable users to deposit collateral into lending contracts on any partner rollup, while market makers borrow against these deposits through t1's cross-chain collateral accounts to fill intents. Loans can be settled and returned to the origin chain within one minute, with a share of bridging fees passed back to depositors as yield. This design reduces capital requirements for borrowers while delivering competitive, real-demand-driven returns to lenders.
- Same-block-deposit-trade-withdrawal (aka *Same-Block-DTW*) dApp flows: A user on Ethereum is able to use t1 to e.g. trade without even realizing they're leaving Ethereum (no wallet network switching etc.). Their original deposit transaction on L1 into t1 may contain e.g. trade intent metadata that gets interpreted by t1 out-of-band and submitted to a marketplace of solvers, resulting in one of them placing a new tx on t1 which is ultimately leading to a withdrawal tx back to the L1—which may be sealed into the very same L1 block as the original deposit tx and succeed or fail L1-atomically.

These use-cases build TVL on t1 and empower the network to become a programmable liquidity layer that any connected chain can tap into. Thanks to RTP, t1 eliminates the

fragmentation of liquidity across isolated ecosystems and enables applications to operate as if liquidity were unified.

t1 is also designed to enhance the functionality of existing applications. For Ethereum L1 applications, t1 offers an execution environment with significantly lower costs while maintaining composability with Ethereum, enabling applications to scale. Appchains and rollups can leverage t1's infrastructure to access cross-chain liquidity without the need for custom integrations or shared sequencer agreements. By being as a permissionless interop layer, t1 ultimately allows partner applications to improve capital efficiency and offer better UX.

Conclusion

Today, applications need to deploy on multiple rollups to meet user demand. In the future, applications built on t1 will be able to serve users across multiple rollups by deploying on t1 alone.

Glossary

See https://docs.t1protocol.com/concepts/glossary.

Acknowledgements

We are grateful to everyone along our journey so far with whom we've discussed or white-boarded ideas for tackling t1's design challenges, often with an intense but productive back and forth—many design choices stem from or were inspired by these valuable interactions. In particular, we'd like to thank Joachim Neu, Noah Citron, Dankrad Feist, Roberto Saltini.