Stater

$STTR

PRE-LAUNCH

One token.
Gas on any chain.

A fixed-supply settlement asset for cross-chain gas, in-app payments and DAO governance, built on Base.

Network Base · chain ID 8453

Supply 10,000,000,000 STTR, fixed · no owner minting

Standards ERC-20 · ERC-2612 · ERC-5805 · ERC-7802

Fees No transfer fee · no freeze · no pause

Summary

Stater ($STTR) is a fixed-supply token on Base that lets a user transact on any blockchain while holding only one asset. Instead of acquiring the native gas token of every network they touch, a user holds STTR and the infrastructure settles gas on their behalf.

The token has three roles: it pays for gas across chains, it is the unit of account inside the Stater social-finance application, and it carries governance rights in the Stater DAO at one vote per token.

STTR is deliberately plain as an asset. It has **no transfer fee**, **no freeze or blacklist**, **no pause switch**, and **no owner minting**. Every one of those mechanisms would break the jobs it has to do — a token used to settle gas and move across bridges has to transfer the exact amount it says it transfers. The contract is built on audited OpenZeppelin components and implements four standards: ERC-20, ERC-2612 for gasless approvals, ERC-5805 for checkpointed voting, and ERC-7802 for native cross-chain movement.

> **Status: pre-launch.** The contract is complete and tested, but it has not been deployed to Base mainnet, has no liquidity, and has not been reviewed by an independent security auditor.

The problem

Every blockchain charges fees in its own native asset. To do anything on Ethereum you need ETH; on Solana, SOL; on Polygon, POL. This is invisible to people already in crypto and close to fatal for everyone else.

The consequences compound:

- **A user cannot spend what they hold.** Someone holding a stablecoin on a chain where they have no native token is stuck. They own value they cannot move, because moving it costs a different asset they do not have.
- **Onboarding to a new chain means starting over.** Acquire the native token first, from an exchange or a bridge, before the first transaction. Every additional chain repeats the friction.
- **Small balances become unusable.** Gas is a fixed cost. A wallet holding a few dollars on three chains may not be able to afford to consolidate them.
- **Applications inherit the problem.** A multi-chain application cannot onboard a user who has no gas anywhere, so it either subsidises them or loses them.

Existing answers are partial. Bridges move assets but still require gas on both sides. Exchanges solve it only for people who already have an account. Gas sponsorship works but usually ties the user to one application, one chain, and one sponsor's balance sheet.

The missing piece is an asset that is accepted as payment for gas **everywhere**, that the user holds once and spends anywhere.

How it works

Three mechanisms combine. None of them is exotic; the design choice is refusing to add anything that would break them.

Approving without gas

STTR implements ERC-2612 `permit()`. A holder authorises a spender by **signing a message**, not by sending a transaction. Signing costs nothing and requires no native token.

This is the piece that makes the rest possible. A user with STTR and no ETH can still authorise a relayer, because the authorisation never touches the chain until someone else submits it.

Settling gas through a paymaster

Under ERC-4337 account abstraction, a paymaster pays a transaction's gas in the native asset and takes payment from the user in an ERC-20. The user signs; the paymaster settles.

Paymasters require the amount they pull to equal the amount they receive. A token that skims a fee on transfer causes validation and execution to disagree, and the operation reverts. **This is the specific reason STTR has no transfer fee.** It is not a marketing decision, it is a functional requirement: a taxed token cannot be used to pay for gas.

Moving between chains without wrapping

STTR implements ERC-7802. Moving between chains **burns** the tokens on the source chain and **mints** the same amount on the destination.

The practical effect is that global supply never changes — only its distribution across chains does. There is no wrapped derivative, no separate liquidity pool per chain, and no custodial pot of locked collateral to be drained. Both functions can only be called by bridge contracts the DAO has explicitly authorised.

Token overview

Name / Symbol	Stater / STTR
Decimals	`18`
Total supply	`10,000,000,000 STTR, fixed`
Home network	Base (chain ID 8453)
Transfer fee	None — on buys, sells and transfers alike
Freeze / blacklist / pause	None
Owner minting	None
Standards	ERC-20, ERC-2612, ERC-5805, ERC-7802

Supply is minted once at deployment and the contract contains no function that can create more. The only way supply moves is downward, through holders burning their own tokens, or sideways between chains through the bridge burn-and-mint mechanism, which conserves the global total.

Base was chosen for three reasons: it is an OP Stack chain inside the Superchain, which gives STTR native cross-chain movement through the canonical bridge rather than a third-party wrapper; it has deep existing liquidity and full Uniswap deployment; and its fees are low enough that gas settlement is economic at small transaction sizes.

Token distribution

The full 10,000,000,000 STTR supply is allocated as follows.

ALLOCATION	SHARE	TOKENS	PURPOSE
Community airdrop and activities	20%	2,000,000,000	Distribution to users, campaigns, rewards for participation
Development	20%	2,000,000,000	Protocol engineering, infrastructure, audits
Grants and independent developers	20%	2,000,000,000	Funding third-party builders on Stater infrastructure
Liquidity	15%	1,500,000,000	DEX liquidity provision and market depth
Team	14%	1,400,000,000	Three founding partners, split 60/20/20
Marketing	10%	1,000,000,000	Growth, partnerships, exchange listings
Early investor	1%	100,000,000	Single pre-launch backer
Total	**100%**	**10,000,000,000**	

Vesting

The team allocation is split across three founding partners at 60/20/20 — 840,000,000, 280,000,000 and 280,000,000 STTR — and each tranche is **locked and released at 5% per month**, taking 20 months to unlock in full. The early investor receives **5% at the start** and the remaining 95% at the same 5%-per-month rate, completing in 19 months.

ALLOCATION	SHARE	RELEASE	FULLY UNLOCKED
Team — partner 1	8.4%	5% per month, linear	20 months
Team — partner 2	2.8%	5% per month, linear	20 months
Team — partner 3	2.8%	5% per month, linear	20 months
Early investor	1%	5% at the start, then 5% per month	19 months

Combined, 15% of supply follows this schedule. The only part liquid at launch is the investor’s opening tranche — 5,000,000 STTR, 0.05% of supply. Roughly 60% of the scheduled total has released by month twelve, with the remainder following through month twenty.

The schedule is stated here as the project's commitment. It becomes verifiable once the vesting contracts are deployed and their addresses published — until then a reader should treat it as intent rather than an on-chain guarantee. The date the schedule starts from has not been fixed.

Reading this honestly

Forty percent of supply — community airdrop plus grants — is directed outward, to users and to developers who are not the core team. Combined team and early-investor allocation is 15%, which is modest by the standards of the category.

That said, an allocation table is a statement of intent, not a guarantee. What converts it into a commitment is **on-chain enforcement**: vesting contracts for the team and investor tranches, published addresses for each bucket, and a liquidity lock with a stated duration.

Not yet defined

The following are unresolved at the time of writing, and a reader should treat them as open:

- **Vesting start date.** The 5% monthly release is set, but the date it begins from has not been fixed.
- **Mainnet addresses.** The vesting contract, the liquidity lock and the six multisigs exist and have been exercised on Base Sepolia, but nothing is deployed to mainnet. Until those addresses are published, the custody model above is a design rather than a verifiable fact.
- **Source verification.** The current contracts have not been verified on a public explorer.
- **Airdrop mechanics.** Eligibility, timing and claim process for the community allocation are not yet specified.

These are the details that determine whether the distribution above is meaningful. They should be settled and published before any public sale or listing.

Utility

STTR has three distinct roles. Each is a reason to hold it that does not depend on the others.

Universal gas

A user holding STTR can transact on any supported chain without holding that chain's native asset. They sign; a paymaster settles the gas and takes STTR in return. From the user's point of view there is one balance and it works everywhere.

This is the primary demand driver. Gas is consumed continuously by anyone using the network, which ties token demand to actual usage rather than to speculation.

Currency of the Stater application

STTR is the unit of account inside the Stater social-finance application — the asset balances are denominated in and payments are settled in.

The absence of a transfer fee matters here more than anywhere. A social-finance product depends on small, frequent transfers. A three percent cut on every payment would make the product unusable, which is a large part of why the fee was removed during design rather than after launch.

Governance

One token, one vote in the Stater DAO. Holders vote on protocol parameters, treasury allocation, grant funding, and — critically — which bridge contracts are authorised to move supply between chains.

That last power is the most consequential thing the DAO controls, and it is covered in detail in the Governance and Security sections below.

Technical architecture

The contract is a single Solidity file built on OpenZeppelin 5.6.1. Using audited components rather than hand-written equivalents is deliberate: for a project positioned in security infrastructure, "we wrote our own ERC-20" is a liability rather than a credential.

STANDARD	WHAT IT PROVIDES
ERC-20	The base token
ERC-2612 (`permit`)	Approval by signature, with no native token required
ERC-5805 (votes)	Checkpointed balances and delegation for governance
ERC-7802 (crosschain)	Burn-and-mint movement between chains
ERC-20 Burnable	Holders can destroy their own tokens

Deployment model

On the home chain, Base, the full supply is minted once. On any other chain, the contract deploys with **zero supply** — the only tokens that can ever exist there are ones a bridge minted against tokens burned elsewhere. This makes it structurally impossible for a remote deployment to create supply out of nothing.

Cross-chain authorisation

`crosschainMint` and `crosschainBurn` are restricted to holders of `BRIDGE_ROLE`. The role is generic rather than hardcoded to one bridge, so the canonical Superchain bridge, and later a LayerZero or Chainlink CCIP adapter for chains outside the Superchain, can each be authorised without modifying the token.

Two details are worth stating precisely:

- **A bridge cannot burn from an arbitrary wallet.** `crosschainBurn` spends an allowance unless the bridge already holds the tokens. A compromised bridge cannot drain holders who never approved it.
- **`BRIDGE_ROLE` can be revoked but not renounced.** A bridge abandoning its own role mid-transfer would strand supply in flight, so removal is a deliberate act by the admin rather than something a bridge can do to itself.

Compilation

solc 0.8.28, optimiser enabled at 200 runs, EVM target `cancun`. Runtime bytecode: `Stater` 8,768 bytes, `StaterVesting` 4,351 bytes, `StaterLiquidityLock` 4,685 bytes — all well inside the 24,576-byte contract limit.

Security model

The distinction that matters is between what a project *promises* and what its code *prevents*. The following are the second kind — each is verifiable by reading the published source, and none depends on the team's continued good behaviour.

PROPERTY	HOW IT IS ENFORCED
Supply cannot be inflated	No mint function exists. `crosschainMint` is bridge-only and exists solely to restore supply burned elsewhere.
No wallet can be blocked from transacting	There is no blacklist, freeze, pause or transaction limit. Beyond the standard ERC-20 checks, the transfer path contains no condition any privileged account controls.
No fee can be introduced	There is no fee mechanism in the contract to enable.
The admin cannot touch balances	The admin role's entire surface is granting and revoking roles.
A bridge cannot burn without approval	`crosschainBurn` spends an allowance unless the bridge holds the tokens itself.
Bridge rights cannot be silently abandoned	`BRIDGE_ROLE` is revocable by the admin but not renounceable by the holder.

Custody

Every allocation sits in one of three places, and each is a separate address anyone can read on a block explorer.

ALLOCATION	SHARE	HELD BY	WHO CAN MOVE IT
Community airdrop and activities	20%	Its own multisig	2 of 3 signers
Development	20%	Its own multisig	2 of 3 signers
Grants and independent developers	20%	Its own multisig	2 of 3 signers
Liquidity	15%	Multisig, then LP locked 365 days	Nobody until the term ends
Team	14%	Vesting contract	Nobody — it pays itself
Marketing	10%	Its own multisig	2 of 3 signers
Early investor	1%	Vesting contract	Nobody — it pays itself

Separate multisigs rather than one pooled treasury, so each portion's balance and spending history stand on their own. A reader can watch the marketing budget without inferring it from a shared balance.

The admin multisig holds `DEFAULT_ADMIN_ROLE` and **no tokens at all**. Compromising it grants no access to any balance.

Liquidity lock

Liquidity provider tokens are deposited into `StaterLiquidityLock` for **365 days**. The contract has no owner, no admin, no pause and no emergency path: nothing in it can release a lock before its date, and no number of signatures changes that. The unlock date can be pushed further out and can never be pulled in, so a published date is a floor rather than an intention. Both LP shapes on Base are supported — ERC-20 pair tokens and ERC-721 concentrated-liquidity positions.

Vesting needs no signature

The team and investor schedules are deliberately outside multisig control. `release()` is permissionless and pays only the beneficiary, so tokens reach the assigned wallets on schedule whether or not any signer is available, willing, or still involved. The multisig cannot speed it up, delay it, or redirect it; after sealing it cannot even recover the vested token.

Who holds the admin role

`DEFAULT_ADMIN_ROLE` is held by a multisig wallet. Every privileged call — granting or revoking `BRIDGE_ROLE`, and handing the role onward — requires the multisig's signing threshold, so no single key can act alone. The role can be transferred to a DAO governor later without touching the token contract.

The admin surface is deliberately small: it grants and revokes roles and nothing else. It cannot mint, move, freeze or tax a balance, so a compromised admin cannot take anyone's tokens. The worst it can do is grant `BRIDGE_ROLE` to an address that should not have it — which is why that role is revocable.

One consequence worth stating: a multisig is a contract and cannot produce an ECDSA signature, so **ERC-2612 `permit` does not work for a contract-held balance**. The token implements no ERC-1271 fallback. A multisig treasury approves spenders with `approve()` instead. This affects contract holders only; ordinary wallets use `permit` normally.

The trust boundary

There is one, and it should be stated plainly rather than buried: **any contract granted `BRIDGE_ROLE` can mint STTR.** That is inherent to burn-and-mint cross-chain movement — a bridge must be able to recreate on one chain what it destroyed on another.

This makes the authorisation decision the most important one the DAO makes. Three practices follow from it:

- Prefer the canonical Superchain bridge over third-party adapters wherever both are options.
- Review every adapter before authorising it, and publish the review.
- Keep the number of authorised bridges as small as the product allows. Each one is an independent way supply could be created improperly.

Verification

The contract has **95 automated tests** across the three contracts, including tests that assert the *absence* of fee, freeze, pause, mint and clawback functions rather than only testing what exists. Static analysis with Slither returns **zero findings for the token**; it has not been run against the vesting or liquidity-lock contracts. On Base Sepolia the full architecture passed **39 live-chain checks with zero failures**.

Neither is a substitute for an independent audit, which has not been performed.

Governance

The Stater DAO governs the protocol at one vote per token. Voting power is measured by ERC-5805 checkpoints rather than by current balance, which matters more than it sounds.

Why checkpoints

Without them, one-token-one-vote is trivially defeated. An attacker borrows a large balance in a flash loan, votes, and repays — all in a single transaction, at almost no cost. Governance decided by whoever can borrow the most for twelve seconds is not governance.

Checkpoints record voting power at each point in time. A Governor reads `getPastVotes` at a timepoint **before** the proposal existed, so a balance acquired afterwards carries no weight. The token's test suite includes this exact attack: an address acquires five billion STTR after the snapshot and its past voting power still returns zero.

Delegation

Voting power is **opt-in**. Tokens do not vote until they are delegated, including delegating to yourself:

```
delegate(myAddress)     // activate my own voting power
delegate(someoneElse)   // hand my voting power to a delegate; tokens do not move
```

This is a genuine usability hazard. Holders who never delegate have zero influence without realising it, and a DAO whose holders have not delegated will look inactive regardless of how many tokens exist. Activating delegation belongs in the application's onboarding flow, not buried in documentation.

Delegating to someone else transfers voting power only — the tokens never leave the holder's wallet.

What the DAO decides

- Which bridge contracts are authorised to move supply between chains
- Treasury allocation from the development, grants and marketing buckets
- Grant funding for independent developers
- Protocol parameters and supported networks

The token's clock runs on timestamps (`mode=timestamp`), so it pairs with a timestamp-based Governor.

Status

Stated plainly, so no reader has to guess which claims are finished and which are intentions.

ITEM	STATE
Token contract	Complete
Vesting contract	Complete
Liquidity lock contract	Complete
Automated tests	95, zero failures
Static analysis	Slither on the token contract only, zero findings
Full architecture exercised on Base Sepolia	39 checks, zero failures
Multisig custody on testnet	Six Safes deployed and exercised
Source verified on a public explorer	**Not yet for the current deployment**
Independent security audit	**Not started**
Base mainnet deployment	Not started
Liquidity provision on mainnet	Not started
Paymaster integration	Not started
Bridge authorisation	Not started
DAO Governor deployment	Not started

What the testnet run covered

The whole architecture was deployed to Base Sepolia and exercised with real transactions: 39 checks, zero failures. The run covered the full supply mint, the absence of any fee, freeze, pause or mint function in the deployed ABI, a gasless `permit()` signed by a wallet holding no ETH, delegation and the flash-loan defence, ERC-7802 burn-and-mint with global supply conserved, a zero-supply remote deployment, and the limits on the admin role.

Two results are worth singling out. A wallet holding **zero ETH** signed an approval and a relayer moved its tokens — the mechanism the whole gas-abstraction design rests on. And an address that acquired five billion STTR *after* a governance snapshot returned zero past voting power, which is the flash-loan attack the checkpointing exists to defeat.

The source of the testnet deployment is published and verified on a public block explorer, so the code can be read directly rather than taken on description.

Risk factors

These are specific to this project. They are not a substitute for independent diligence.

Vesting is proven on testnet, not on mainnet. The team and early-investor allocations, 15% of supply combined, release at 5% per month; the team over 20 months, the investor over 19 after a 5% opening tranche. The contract has been deployed, funded, sealed and released from on Base Sepolia. On mainnet it does not exist yet, so nothing on-chain constrains those tokens today. It becomes a real constraint once the contract is deployed, sealed and its address published — sealing is refused unless the contract already holds every token it promises.

Allocation is designed and rehearsed, not yet live. The custody model — a separate multisig per bucket, a sealed vesting contract and a liquidity lock — has been deployed end to end on Base Sepolia, with every token accounted for. No mainnet address exists yet, so a reader cannot currently verify the split on the chain that will matter.

No independent audit. The three contracts have 95 automated tests and 39 passing live-chain checks, and the token has a clean static-analysis report, but no third-party security firm has reviewed any of them. Testing shows the code does what its authors intended; it cannot show the absence of flaws they did not consider.

Bridges can mint. Any contract granted `BRIDGE_ROLE` can create STTR on the chain it operates on. A compromised or malicious bridge is the most severe technical risk in the design, and it scales with the number of bridges authorised.

The utility depends on integrations that do not exist yet. Gas abstraction requires paymaster deployments; cross-chain movement requires authorised bridges. Neither is live. Until they are, STTR's stated utility is a plan rather than a working product.

Deployed code is immutable. The contract cannot be patched after deployment. A flaw found later cannot be fixed in place; it would require a migration.

Network risk. Base is a layer-2 with a centralised sequencer. Outages, reorganisations or policy changes at the network level are outside the project's control.

No market exists. The token is not deployed, has no liquidity and no trading history. There is no assurance a liquid market will develop or persist.

Regulatory treatment is undetermined. Depending on jurisdiction and on how the token is marketed and sold, STTR may be treated as a security or another regulated instrument. No determination has been made and no regulatory approval has been obtained anywhere.

Disclaimer

This document describes the design and mechanics of a software contract. It is not an offer to sell or a solicitation to buy any asset, and it is not investment, legal, accounting or tax advice.

Figures in this document describe contract behaviour and stated allocations. They are not projections, forecasts, or representations of future performance. Nothing here should be read as a promise of value, return, listing, or liquidity.

Statements about future plans — integrations, deployments, governance, audits — are intentions at the time of writing, not commitments. They may change or not happen.

Digital assets are volatile and can lose all of their value. Smart contracts can contain flaws that testing and static analysis do not find. Anyone considering acquiring STTR should read the contract source, form their own view, and consult their own advisers.

Compiled with solc 0.8.28, optimiser at 200 runs, EVM target `cancun`. Base mainnet 8453, Base Sepolia 84532.