Universal Gas Framework

Tychi Labs

Yash Singh Rudr Rishi

yash@tychiwallet.com rishi@tychiwallet.com

Version: v1.1.0 (Mainnet)
Nov 2025

Abstract

The Universal Gas Framework (UGF) is a cross-chain transaction execution protocol developed by Tychi Labs that enables users to prepay gas on one blockchain and execute transactions on another. Designed to simplify and unify cross-chain user experiences, UGF abstracts away multi-network gas requirements via a relay layer powered by smart wallets, gas prepayment contracts, and multi-relayer validation. Now live on mainnet, UGF supports EVMs, Solana, Sui and Tron, providing a gas-abstracted gateway to interoperable applications in Tychi Wallet.

Why Cross-Chain Isn't Seamless

Let's say you're a user who primarily operates on Base Chain — you trade there, hold assets, and it's where most of your gas balance sits. But over time, you start interacting with other ecosystems. A client pays you on Solana. Airdrops or DeFi opportunities appear on Ethereum or Sui. You now have assets on multiple chains — but they're stuck.

To actually use your tokens, two major problems emerge:

- **You need to buy native gas on each chain — just to make a transaction.** Even if you already hold tokens on Solana or Ethereum, you can't move or trade them unless you first acquire SOL or ETH separately, often through a centralized exchange or bridge.
- **Your liquidity is fragmented across chains, and accessing it requires bridging or wrapping.** Want to use an opportunity on one chain? You're forced to bridge your tokens from another chain — an extra step that is slow, costly, and error-prone.

These frictions break the promise of a seamless, interoperable crypto ecosystem and make cross-chain participation needlessly complicated.

Introducing Universal Gas Framework

The Universal Gas Framework (UGF) is the core infrastructure layer for multi-chain capabilities. It is designed to solve one of the most persistent UX bottlenecks in blockchain: fragmented gas fees across networks.

With UGF, users no longer need to acquire and manage native gas tokens on every chain they interact with. Instead, they hold **ETH on Base**—their single, low-cost settlement layer for every supported network. Whether the user wants to execute on Ethereum, BNB Chain, Solana (for SPL token transfers), or Sui (for sponsored Move contract calls), UGF handles the complexity of gas forwarding and execution beneath the surface.

UGF v1.1.0 (Mainnet) supports:

- EVM chains (e.g., Ethereum, Polygon, Arbitrum, Base, BNB Chain, Avalanche, Optimism)
- Solana
- Sui
- TRON (Energy Bandwidth delegation)

A public UGF SDK is already available, allowing third-party wallets and dApps to integrate gasless execution directly.

UGF is the invisible layer that lets users pay once and execute anywhere — bringing true interoperability to life.

Design Philosophy: Why We Built UGF This Way

The Universal Gas Framework (UGF) was designed with a single premise in mind: users shouldn't have to think about gas — especially not across chains. In today's fragmented Web3 ecosystem, every blockchain demands its own native gas token. This requirement disrupts user experience at a fundamental level. Sending tokens on Ethereum requires ETH. Claiming an airdrop on Solana requires SOL. Even simple DeFi actions often involve manual gas top-ups through CEX withdrawals or cross-chain bridges. UGF was built to eliminate this friction entirely.

Our goal was to make cross-chain interaction feel seamless — not like a series of disconnected steps. We envisioned a system where users could hold gas on a single chain, pay once, and execute transactions on any supported network. To accomplish this, we introduced a number of foundational components and principles.

On EVM-based chains, UGF supports both per-user smart wallets (enabling gasless transactions, signature validation, and nonce management) and EOA funding, where UGF directly transfers the required native gas to user accounts when needed. For Solana and Sui — which currently operate with EOAs only — the v1.1.0 release introduces full sponsorship support, including both direct SPL/Move transaction signing and direct funding of user wallets with the necessary gas. To securely manage gas prepayment, FuelStation contracts are deployed on the designated settlement layer, with Base chain serving as the primary chain in v1.1.0.

Rather than using traditional bridging mechanisms, UGF leverages a decentralized network of relayers that independently validate and co-sign each transaction. This multi-relayer consensus model enhances both security and execution speed, enabling near real-time cross-chain interactions. The system is designed to be chain-agnostic from the ground up. While the initial release supports EVM chains, Solana (SPL transfers), and Sui, the architecture is modular enough to onboard chains like Aptos, XRP, and even Bitcoin in later phases.

UGF is built for developers from day one. SDK and API suite are already live, enabling seamless gas abstraction across wallets, dApps, and platforms. Going forward, our focus is on continuous expansion — integrating more chains, strengthening infrastructure, and evolving UGF into a unified cross-chain execution layer.

UGFSCAN (https://ugfscan.com) is fully operational as UGF's cross-chain transaction explorer and will continue to advance as the network grows.

With UGF, we deliver a unified, gas-abstracted user experience — one where users can prepay once on a settlement layer, execute transactions across multiple blockchains, and never be blocked by fragmented gas infrastructure again.

Why Base and ETH?

UGF originally launched using BNB on opBNB as its settlement layer, but we migrated to **Base ETH** due to its superior stability, developer ecosystem, and rapidly growing user adoption. As an Optimism-based Layer 2, Base provides extremely low fees, fast confirmations, and a reliable environment for high-throughput cross-chain gas abstraction.

Users now only need to hold **ETH on Base** as their universal gas balance. All UGF Gas payments, job validations, and relayer coordination occur on Base, making it the global omnichain settlement layer for UGF v1.1.0.

UGF SDK Integration

To extend UGF beyond Tychi Wallet, a lightweight public SDK is available for any wallet, dApp, custodial service, or backend infrastructure that wants to offer unified, gas-abstracted execution. The SDK exposes a minimal surface:

- prepare a transaction (EVM, Solana, Sui, TRON)
- receive a Base-ETH or TYI gas quote
- pay Gas on Base
- let UGF handle relayer consensus, resource delegation, and execution

Developers do not manage relayers, RPC nodes, simulation logic, Sui sponsorship, or TRON resource delegation. The SDK abstracts all of it.

Omnichain Settlement Layer (Base). UGF uses Base as its global omnichain settlement layer. All gas payments and job validations occur on Base, allowing any application to execute cross-chain actions while maintaining a single gas balance.

Core Capabilities:

- Unified API for EVM, Solana, Sui, and TRON
- Automatic digest creation from calldata or serialized instructions
- Accurate Base-ETH/TYI quoting for any destination chain
- Helpers to pay gas to the UGF FuelStation on Base
- Automatic relayer quorum handling

Who Can Integrate:

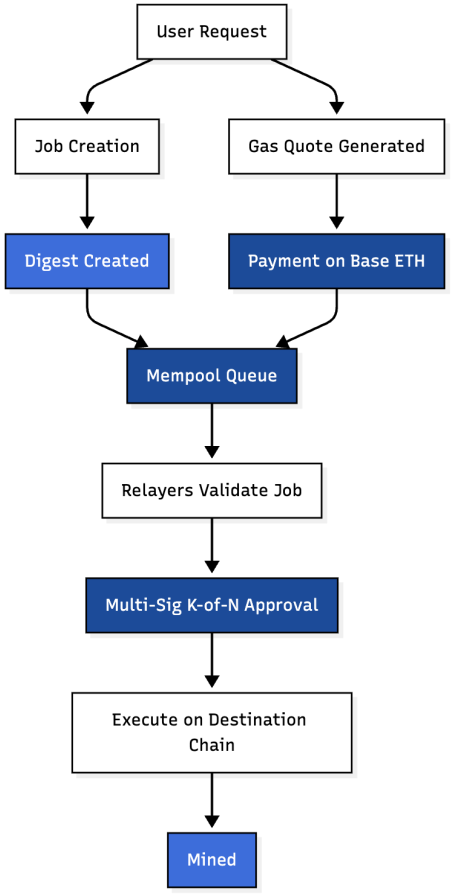
- Mobile and extension wallets
- DeFi and GameFi dApps
- Trading bots and automation tools
- Custodial platforms and fintech apps
- Backend microservices requiring cross-chain execution

Developer Experience (DX). Most integrations require only three calls:

1. `getQuote()` — prepare job and receive Base-ETH price
2. `payFuel()` — user pays gas on Base
3. `execute()` — UGF finalizes execution or sponsorship

Documentation https://universalgasframework.com/
Integration takes minutes, enabling any app to offer gasless, chain-agnostic execution without users holding multiple native tokens.

System Architecture: How UGF Coordinates Gasless Cross-Chain Execution



0.1 User Request and Job Creation

The gasless transaction flow begins when a user initiates a cross-chain request by specifying the destination chain and providing the raw transaction details. UGF parses this request and constructs a chain-specific job object, caching it temporarily with a unique digest to prevent replay attacks and to track payment status.

EVM Chains For EVM-compatible chains (e.g., Ethereum, BNB Chain), the job includes the following fields, which define the target contract, input data, gas cost quote, and user metadata:

- `from`: the externally owned account (EOA) initiating the request
- `to`: the destination contract address to interact with
- `callData`: ABI-encoded function and parameters
- `chainId`: numeric chain ID of the target chain

- `quoteWei`: estimated gas fee (in wei) for the transaction
- `quoteInETH`: the equivalent ETH value to be paid on Base
- `userNonce`: optional user-supplied nonce (used for replay protection or ordering)
- `validUntil`: expiration timestamp of the quote in milliseconds
- `status`: the current lifecycle status of the job (e.g., `quoted`, `paid`)

These fields are cached in Redis using a unique digest key.

Solana The job captures a base64-encoded serialized Solana transaction (`transactionInstructionBase64`), which is later deserialized and dry-run to estimate compute units.

Sui For Sui, the job includes transaction metadata with a unique digest computed using BLAKE2b hash of the transaction bytes, random salt, and timestamp. This digest serves as a unique identifier across chains and the job is stored temporarily with expiration controls. The job object is ephemeral and expires if the user does not confirm and pay within a defined TTL. Once paid, the backend marks it as valid and ready for relayer coordination.

TRON For TRON, the job stores the raw serialized transaction (`rawTxHex`) with the chain ID and basic metadata required for sponsorship. The backend simulates the transaction to estimate the Energy and Bandwidth needed, converts this into a TRX cost, and then into a Base–ETH quote for the user. The job is saved under a unique digest and proceeds to relayer approval once the Base–ETH payment is made.

0.2 Gas Estimation and Quote Delivery

Once the job object is created, the backend estimates the transaction cost on the destination chain. This process varies per chain type:

EVMs

The backend uses on-chain calls (such as `eth_estimateGas`) to simulate the transaction and compute the expected gas usage. This value is then multiplied by the current network gas price to derive the total cost in wei.

Solana

For Solana, the serialized base64 instruction is deserialized and dry-run via `simulateTransaction`. The resulting compute unit estimate is mapped to a lamport cost, which is then converted to ETH.

Sui

The Sui backend uses `devInspectTransactionBlock` to simulate the transaction. The result includes a breakdown of computation and storage costs, which are combined into a native gas fee estimate.

Tron

For TRON, the backend determines the required **Bandwidth** and **Energy** for the user's transaction, based on TRON's resource model. These resource requirements are priced in TRX and then converted into a Base-ETH–equivalent quote. This quote reflects the cost for UGF to delegate the necessary Energy and Bandwidth to the user's address for execution.

Quote Conversion

The estimated native gas cost is then converted into a Base-ETH–equivalent quote using real-time price feeds (e.g., Chainlink, CMC, or backend-signed conversion rates). This ETH quote is returned to the client along with the unique digest and expiry timestamp. The quote is now awaiting user payment on Base.

Cost Conversion We convert the estimated gas units into ETH and add up to a 10% slippage:

$$\text{Cost}_{\text{ETH}} = \text{gasUnits} \times \text{gasPrice} \times \frac{\text{ETH}}{\text{nativeToken}} \times (1 + 0.10)$$

0.3 EOA Gas Funding

For chains using EOAs, UGF implements direct gas funding. After relayer consensus, UGF transfers native gas tokens directly to the user's EOA, allowing them to broadcast their transaction independently.

Pragmatic Multi-Modal Approach UGF implements a bridge solution that delivers gas abstraction across all networks today, regardless of account abstraction support. Rather than waiting for universal smart wallet adoption, we maintain UX consistency through adaptive execution modes—smart wallets where available, direct funding elsewhere.

Phased Migration Strategy **EIP-7702 Migration (2026-2027)** Gradual transition to native account abstraction as major chains implement EIP-7702.

0.4 Gas Payment on BASE

To proceed with execution, the user pays the quoted ETH amount to the UGF FuelStation contract deployed on **Base**. This payment acts as a cross-chain gas deposit and is linked to the previously issued digest.

```
function payForFuel(bytes32 digest) external payable nonReentrant {
        require(paid[digest] == 0, "FS : Zero Value");
        uint256 amount = msg.value;
        paid[digest] = amount;
        owner.transfer(amount);
        emit FuelPaid(msg.sender, digest, amount);
    }
```

Listing 1: FuelStation Payment Handler

Upon detecting a valid payment event, the FuelStation emits an on-chain event with the associated `digest`, amount, and sender address. UGF relayers listen to these events in real time. Once the payment is confirmed and matches the quoted amount, the job is marked as `ready` in Redis, making it eligible for multi-relayer signing and execution on the destination chain.

This payment mechanism ensures:

- **Replay protection**: Each digest is unique and expires if unpaid.
- **Chain-agnostic flow**: Payment occurs on the Base settlement layer.
- **Fair metering**: Users only pay after receiving an accurate gas quote.

0.5 Distributed Mempool Validation

After a successful gas payment, the associated transaction digest is marked as `ready` in Redis. This state change activates UGF's distributed validation layer — a decentralized coordination mechanism where multiple relayers independently monitor, verify, and act upon eligible jobs.

UGF operates multiple **relayer agents** — stateless, containerized services (e.g., Docker/Kubernetes). These agents (hereafter simply "relayers") are not blockchain nodes but off-chain services that subscribe to job status events and co-sign digests.

The term **distributed** refers to the architectural design: instead of relying on a single centralized processor, UGF deploys multiple relayer agents (hereafter simply "relayers") across different geographic regions. These stateless services subscribe to job status events and Redis keyspace changes in real time, enabling scalable and fault-tolerant processing.

Each relayer performs lightweight validation checks:

- Confirms that the job is marked `ready` and has not expired.
- Validates that no duplicate submission is in-flight for the same digest.
- Prepares the transaction for the final k-of-n consensus signing step.

Redis acts as a shared coordination bus and lock manager, ensuring that even in a distributed setup, job contention and double execution are prevented. Relayers compete to acquire short-lived distributed locks per digest before proceeding to the next phase.

This approach ensures high availability, horizontal scalability, and tolerance against relayer downtime — making UGF resilient under heavy load or partial relayer agent failures.

0.6 Multisignature Consensus: k-of-n Relayer Signing

In UGF v1.1.0, each job requires a k-of-n relayer signature quorum. Out of the total n relayers, at least k must sign the job digest before it can proceed to execution. This ensures that no single relayer can act alone, while the system remains tolerant to failures or downtime.

Expected Load Since only k signatures are needed, not every relayer must participate in every job. On average, each relayer contributes to about

$$\text{Workload}_j \approx \frac{k}{n} \times \text{TotalJobs}$$

signing events. This spreads the workload evenly and prevents bottlenecks.

Signing Flow

1. Each relayer independently verifies the digest D_i.
2. If valid, it signs:
$$\sigma_j = \texttt{Sign}_{sk_j}(D_i).$$
3. Once k distinct valid signatures are collected, the job status is atomically upgraded to `approved`.
4. Any relayer may then forward the approved job to the destination chain.

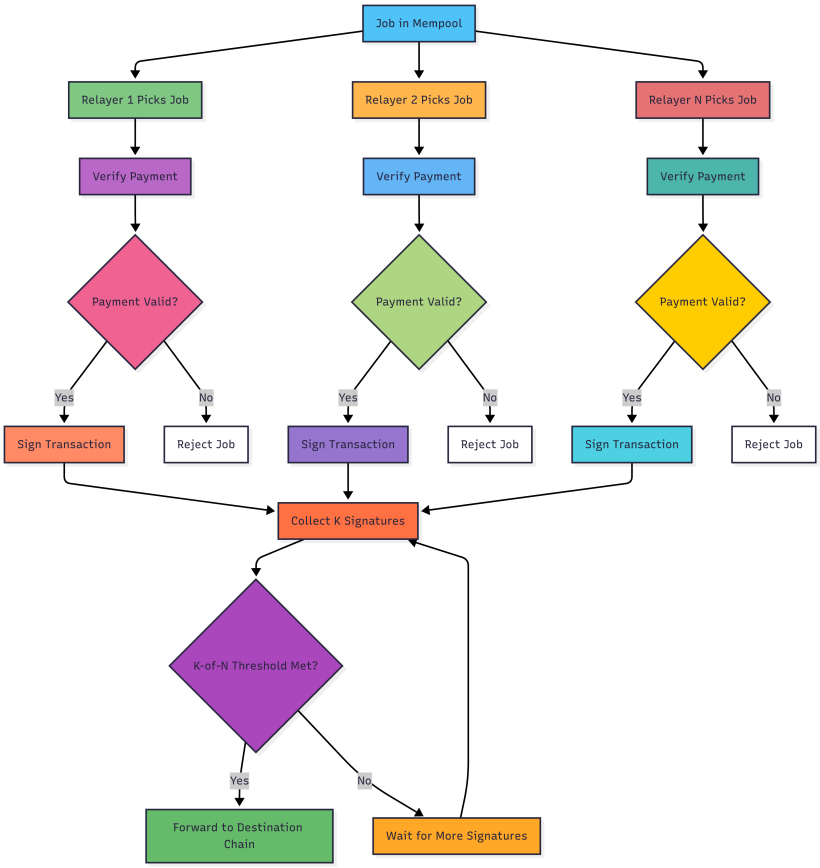
This model ensures:

- **Minimal trust:** No single relayer can unilaterally execute a job.
- **Scalability:** Signatures are gathered asynchronously, so the system grows with the number of relayers.
- **Fault tolerance:** The system tolerates up to $n - k$ offline or malicious relayers without disruption.

Parameterization & Future Enhancements UGF's signature threshold k and total relayers n are configurable off-chain parameters. Future enhancements may include stake-weighted selection, and slashing-based accountability for production validator sets.

Benefits:

- **Byzantine fault tolerance (BFT)** in adversarial conditions.
- **Decentralized control** across larger validator sets.
- **Improved redundancy** across geographic and infrastructure diversity.


Job in Mempool
Relayer 1 Picks Job
Relayer 2 Picks Job
Relayer N Picks Job
Verify Payment
Verify Payment
Verify Payment
Payment Valid?
Payment Valid?
Payment Valid?
Yes
No
Yes
No
Yes
No
Sign Transaction
Reject Job
Sign Transaction
Reject Job
Sign Transaction
Reject Job
Collect K Signatures
K-of-N Threshold Met?
Yes
No
Forward to Destination Chain
Wait for More Signatures

0.7 Destination Chain Execution

Once a job digest reaches the `approved` status — meaning it has been signed by the required set of relayers — it is eligible for submission to the destination blockchain.

EVM Chains

For EVM-compatible chains, UGF finalizes the transaction by calling the on-chain execution function of a trusted smart wallet or execution contract. This function typically validates the original user's signature and forwards the call to the target contract. An example implementation is shown below:

```
function executeOp(
    address _to,
    uint256 _value,
    bytes calldata _data,
    bytes calldata _usersignature
) external returns (bytes memory) {
    bytes32 _operationhash = keccak256(
        abi.encodePacked(_to, _value, _data)
    );

    uint256 validationStatus = _validateSignature(
        _operationhash,
        _usersignature
    );
    require(validationStatus == SIG_VALIDATION_SUCCESS,
        "UGF: invalid user operation");

    (bool success, bytes memory returnData) = _to.call{value: _value}(
    _data);
    require(success, "UGF: execution failed");

    return returnData;
}
```

Listing 2: EVM Smart Execution

This pattern ensures that only operations properly signed by the original user (and quoted in the UGF flow) are allowed to reach execution. Any tampering with parameters will invalidate the hash and revert the transaction.

Solana

UGF reconstructs the original Solana instruction set, signs it with the designated relayer keypair, and submits it via the JSON-RPC 'sendTransaction' endpoint. The transaction confirmation is tracked via polling or websockets, and Redis is updated accordingly.

Sui (User-Broadcasted)

Unlike EVM and Solana, the Sui job is not submitted by UGF. Instead, after receiving UGF's sponsored gas signature (step shown earlier), the user constructs the full transaction locally and submits it to the Sui network. UGF monitors its completion via the Sui client and marks the job as `completed` upon success.

TRON (Resource Delegation Model)

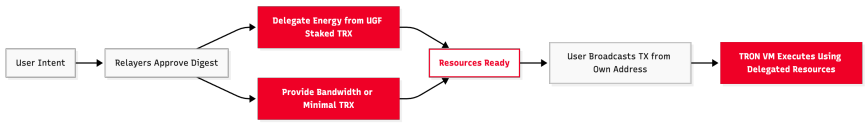
Unlike EVM, Solana, or Sui, TRON introduces a dual–resource execution model based on **Bandwidth** and **Energy**. Every TRON transaction consumes Bandwidth for raw transaction bytes, and smart-contract execution additionally requires Energy. Both resources are funded through TRX, forcing users to hold native TRX even if their intent is solely to transfer TRC-20 tokens such as USDT. UGF removes this dependency by abstracting TRON's resource mechanics and providing

execution through delegated resources sourced from internally staked TRX.

Bandwidth. Bandwidth is a byte-based resource consumed for every transaction broadcast, regardless of contract interaction. Simple transfers may require only Bandwidth, whereas contract calls consume both Bandwidth and Energy. If the user has no staked TRX on TRON, Bandwidth is consumed by burning TRX directly from their balance.
Energy. Energy is consumed during smart-contract execution and can be supplied either from staked TRX (regenerating Energy) or by directly burning TRX. If an account has no Energy, the TRON VM automatically burns TRX to cover the required execution cost; execution only fails when neither Energy nor sufficient TRX is available.

Staking and Resource Delegation. UGF maintains a pool of staked TRX across internal operational accounts and partner resource providers. This staked TRX continually regenerates Energy, which can be delegated on-chain to a user's TRON address for temporary execution. TRON's delegation model allows an account to allocate its regenerating Energy to another account's transaction without transferring ownership or custody of tokens. Bandwidth may be provided either through implicit delegation (if sufficient Bandwidth is available) or through a minimal TRX attachment to the user's transaction to cover byte-level costs.

This delegated approach allows UGF to supply the exact Energy required for a sponsored transaction. The Energy consumed during execution is deducted from UGF's staked TRX pool, and naturally regenerates over time on the delegator's account. Users never receive TRX or custody of any resources; UGF only provides temporary execution capacity to complete the transaction.



UGF TRON Resource Delegation Flow

UGF Resource Sponsorship Flow. When a user submits a TRON transaction through UGF and settles the cross-chain fee on Base (or with TYI), the job proceeds through the standard UGF relayer pipeline. Once the relayers reach the k-of-n signature quorum and approve the digest:

- UGF allocates Energy to the user's TRON address from its staked TRX pool.
- If required, UGF attaches a small amount of Bandwidth (via delegated Bandwidth or minimal TRX funding).
- Once resource delegation is confirmed, **the user broadcasts the transaction from their own TRON address**, and the required Energy and Bandwidth are consumed from UGF's delegated resource pool rather than from the user's TRX balance.
- Note: UGF never signs TRON transactions. The user always signs and broadcasts the final TRX transaction from their own address; UGF only supplies delegated resources.

This process enables three TRON execution modes within UGF:

1. **Full Sponsorship**: UGF provides both Energy and Bandwidth for users with zero TRX.
2. **Bandwidth-Only**: If the user has sufficient Energy (e.g., they previously staked TRX or hold contract-granted Energy), UGF supplies only Bandwidth.

By combining TRX staking, delegated Energy, and byte-level Bandwidth provisioning with UGF's cross-chain payment and relayer consensus pipeline, TRON transactions become fully gas-abstracted without requiring users to acquire TRX or understand the underlying resource model.

Finalization Checks

Before marking any job as `completed`, the relayer verifies:

- The digest still matches the signed payload.
- The transaction receipt (if available) indicates success.
- No duplicate submission has occurred.

This ensures proper execution across heterogeneous chains with minimal trust and full off-chain coordination.

0.8 Job Lifecycle Summary

The complete lifecycle of a gasless cross-chain job in UGF spans seven coordinated phases:

1. **User Request**: The user submits transaction intent (to, calldata, chain, etc.).
2. **Gas Estimation & Quote**: Backend performs dry-run on target chain and issues a ETH quote along with a digest.
3. **Gas Payment**: The user transfers ETH to UGF's FuelStation on Base, tied to the digest.
4. **Distributed Validation**: All relayers detect the job status change to `ready` and verify eligibility.
5. **k-of-n Consensus Signing**: Relayers verify and sign the digest; once k valid signatures are present in Redis, the job is approved.
6. **Execution**: : Upon reaching the signature threshold, UGF performs chain-specific execution. For EVM and Solana, a relayer submits the transaction; for TRON, the user broadcasts the transaction after resources are delegated; and for Sui, the user submits the signed transaction locally.
7. **Finalization**: The backend confirms transaction success, marks the job `completed`, and emits optional posthooks or events.

This pipeline enables UGF to abstract gas payments, support multi-chain destinations, and maintain a decentralized relayer mesh — ensuring security, reliability, and scale.

0.9 Security Model and Guarantees

UGF enforces a multi-step verification process to ensure cross-chain job integrity, payment authenticity, and signer accountability.

1. Digest-Based Uniqueness and Expiry Each job is identified by a digest derived from the transaction payload, salt, and timestamp. This prevents reuse and guarantees time-limited validity. Jobs expire if the gas is not paid within the allowed TTL.

2. Signature Verification

- **User Signatures** are verified on-chain before transaction execution (EVM) or before gas sponsorship (Sui).
- **Relayer Signatures** follow a defined k-of-n quorum. Only digests with the required number of valid relayer signatures proceed to execution.

3. Distributed Relayer Checks All relayers independently watch for eligible jobs. Before taking action, each relayer checks:

- If the job is marked as ready
- If a payment exists for the exact quoted amount
- If no one else is already processing the same job

This ensures no job is double-processed or skipped.

4. Controlled Gas Payment The on-chain FuelStation contract ensures that:

- A digest can only be paid once
- The value sent exactly matches the backend quote

5. Stateless and Isolated Relayers Each relayer runs as a containerized service with no persistent state. If one fails, others continue operating without interruption.

6. Full Job Audit Trail Each step — quoting, payment, signing, and execution — is logged with:

- User address, chain ID, and digest
- Gas quote and payment timestamp
- Relayer signature details
- Final transaction hash

7. Adjustable Signing Threshold UGF allows updating the relayer set size n and required signatures k before finalization. This helps adapt to network conditions and validator trust levels.

0.10 Rate Limiting and Abuse Prevention

UGF enforces strict abuse protection to ensure reliability and fair usage; certain controls are being continuously refined and rolled out.

Per-User and Network Limits Each user and network is limited in terms of:

- Quote requests per minute
- Unpaid job count
- Sponsored signature attempts

These are tracked via in-memory counters and Redis TTLs.

Quote Expiry Unpaid quotes auto-expire after a fixed TTL to prevent stale or abandoned jobs from clogging the system. Each quote auto-expires after TTL:

$$\text{expiresAt} = \text{issuedAt} + \text{TTL}$$

Replay and Spam Filtering Duplicate submissions and replayed digests are blocked. Abnormal patterns — like rapid replays or invalid signature bursts — lead to temporary suspensions.

Security Threats and Mitigations To ensure the integrity and robustness of the system, UGF employs several security measures against potential attacks. Below is a summary of key threats and their respective mitigations:

Threat	Mitigation
Replay attack	Unique digest + TTL
Single-relayer compromise	k-of-n signing threshold
Denial-of-service	Rate limits

Table 1: Security Threats and Mitigations

Future Dynamic Controls Adaptive throttling will continue to evolve based on relayer bandwidth, network load, and user reputation.

0.11 Future Roadmap and Expansion

After the successful mainnet launch of UGF v1.0, several upgrades are planned to enhance performance, flexibility, and decentralization:

- **Validator Model (v2.0.0)**: A future version will introduce peer-to-peer validators in a decentralized mesh, removing the need for centralized relayers.
- **Own Blockchain Layer**: UGF v2.0 will include its own consensus layer to coordinate validators, validate jobs, and serve as the universal execution rail.
- Future versions may explore introducing a lightweight coordination layer or dedicated validator mesh, depending on network demand and ecosystem growth.

Supported Networks in v1.1.0 (Mainnet)

Network Type	Supported Chains
Solana	Solana Mainnet (SPL token transfers)
EVM-Compatible	Ethereum, Polygon, Arbitrum, Base, BNB Chain, Avalanche, Optimism
Sui	Sui Mainnet (Sponsored gas via UGF)
TRON	TRON Mainnet (Energy & Bandwidth Delegation)

Table 2: Blockchains supported by UGF in v1.1.0

Conclusion

The **Universal Gas Framework (UGF)** represents a significant advancement in the evolution of cross-chain interoperability. By leveraging **ETH on Base** as a unified gas currency and integrating a **distributed relayer consensus** model, UGF seamlessly abstracts away the complexities of managing native gas tokens across multiple blockchains. This not only simplifies the user experience but also facilitates more efficient and secure transactions across disparate blockchain ecosystems.

Through the introduction of **gasless transactions**, UGF removes the need for users to acquire native tokens on each individual chain, enabling them to focus on their blockchain interactions without worrying about the underlying complexities of gas management. UGF's architecture, built on scalability, security, and modularity, lays the foundation for future cross-chain innovation, making it a cornerstone in the development of decentralized applications (dApps) and blockchain services.

Looking ahead, the **UGF v1.1** roadmap includes support for additional chains, developer SDKs for easy integration, and the potential for transitioning to a **validator-driven consensus model**, ensuring the protocol's robustness and decentralization at scale. As the ecosystem matures, **UGF** aims to redefine the landscape of cross-chain transaction management, setting a new standard for **frictionless, gas-abstracted interactions**.

We invite developers, blockchain projects, and early adopters to engage with the **Tychi Protocol**, contribute feedback, and explore its potential to revolutionize cross-chain interoperability. The future of a unified blockchain ecosystem is within reach, and **UGF** is at the forefront of this transformation.

References

[1] Ethereum Foundation, *Ethereum Whitepaper*, 2013, https://ethereum.org/en/whitepaper/

[2] Base Foundation, *Base: An Ethereum L2 Built on the OP Stack*, 2023, https://base.org/

[3] Solana Labs, *Solana: A New Architecture for a High-Performance Blockchain*, 2020, https://solana.com/

[4] TRON DAO, *TRON Developer Documentation: Energy and Bandwidth Model*, 2021, https://developers.tron.network/

[5] Mysten Labs, *Sui JavaScript SDK*, 2022, https://sdk.mystenlabs.com/

[6] Tendermint, *Tendermint: Consensus without Mining*, 2018, https://tendermint.com/static/docs/tendermint.pdf

[7] Docker Inc., *Docker: Empowering App Development for Developers*, 2021, https://www.docker.com/