THIS DOCUMENT IS AN UNFINISHED DRAFT

Contract Level Compliance

Whitepaper V1

contractlevel.com | github.com/contractlevel/compliance

Abstract

Contract Level Compliance (CLC) is a framework for implementing regulatory compliance in smart contracts. Built to address the inevitable demand for identification of onchain actors in accordance with global regulations, CLC enables protocols to securely, privately, and efficiently verify a user's compliant status before executing restricted logic on their behalf.

By integrating the *Everest identity oracle* [1], *Chainlink Automation* [2], and *Concero V2* [3], Contract Level Compliance ensures that protocols across chains can confidently execute regulatory-compliant smart contracts, whilst maintaining the highest degree of decentralisation possible.

This new generation of regulatory-compliant smart contracts will enable the creation of public chain applications that any onchain actor *and* traditional institution can use, marking an industry first in the combination of both markets.

1. Introduction

The era of blockchain and DeFi as entirely unregulated global markets are gradually, but firmly coming to an end. Regulators across the world have already begun introducing guidelines and enforcement measures for protocols and service providers in the industry [4]. The EU's MiCA regulation is law [5] and the US president signed an executive order to provide regulatory clarity for blockchain and digital assets [6].

In the real world, people are required to disclose certain information about themselves to be able to use traditional financial services [7] and engage in other regulated activities, such as buying and selling property. Centralised exchanges are already required to collect KYC data on their customers [8]. Onchain protocols will be no different - onchain actors will require identification [9].

Regulators will not accept KYC data from anyone. They require the entity facilitating the KYC to enact certain measures including, but not limited to, ongoing monitoring and risk rating [10] - activity that most smart contract based projects do not have the time, resources, or desire to perform.

As smart contracts and blockchains are built upon the foundational idea of decentralisation [11], this poses a unique challenge. Regulatory compliance relies on centralised oversight and trusted intermediaries, whereas onchain protocols function autonomously, without a governing authority to enforce identity verification or risk assessments. This structural

incompatibility makes it difficult to implement compliance without introducing centralisation risks or compromising the permissionless nature of blockchain networks. As a result, Contract Level Compliance must strike a balance—ensuring regulatory adherence while preserving decentralisation.

Currently the only option for striking this balance and *publicly* implementing compliance at the contract level is with Chainlink, the industry standard for offchain computation [12], and Everest, a licensed data provider to the Chainlink network [13].

2. Architectural Overview

The core functional objective of Contract Level Compliance is to make executing compliant smart contracts as simple as possible. To achieve this, CLC is built upon four core principles that define its design:

- **Security** – The system must function securely to ensure smart contracts behave as intended (see section on *Security* for more details).
- **Decentralisation** - The Chainlink network must be used for offchain communication (requests for identity data) and computation (automated response handling) because it provides the securest way of performing these activities without compromising decentralisation.
- **Privacy Protection** - No information about the user must be revealed other than a simple "yes" or "no" response to the question: "Has this address completed KYC with a regulated entity?”
- **Legal Compliance** - Identity data must be sybil-resistant to ensure it cannot be transferred in blockchain environments, preventing money laundering and identity fraud. Additionally, the data must be issued by a licensed and regulated entity to maintain compliance.

To achieve these design principles, the system relies on three key integrations that form the foundation upon which the *Router* and *Logic* contracts operate.

2.1 Integrations

Everest, Chainlink, and Concero each play an important role in ensuring that Contract Level Compliance functions efficiently across chains. Everest provides sybil-resistant identity verification, Chainlink facilitates secure offchain computation, and Concero enables fast, cost-effective crosschain messaging.

2.1.1 Everest

Everest is a regulated provider of sybil-resistant identity data to smart contracts. Everest solves the deduplicated identity problem by using a person’s biometrics as the private key to their identity. This prevents transfer of the identity, ensuring an attacker cannot use someone else’s compliant status. Everest’s identity solution paired with their strict regulatory compliance makes it the ideal solution for implementing Contract Level Compliance.

Users of the Everest platform own their own data and submit KYC information voluntarily. This data is never exposed by the identity oracle, which merely indicates if a user has completed KYC or not.

2.1.2 Chainlink

Chainlink is the industry standard for offchain communication and computation for smart contracts. Chainlink has proven itself as a secure standard for offchain computation, securing most of the DeFi industry since its inception and servicing traditional institutions..

2.1.3 Concero

Concero improves upon Chainlink CCIP by reducing fees and crosschain transaction times, whilst maintaining the security of CCIP as a settlement layer. Concero achieves this with crosschain pools and its own crosschain messaging layer (built with Chainlink Functions and Eigenlayer) for transaction execution.

2.2 Router

The CLC Router contract will be deployed on every Concero Compatible Chain (CCC) by the contractlevel.com team. This contract will be used to make requests for the compliance status of an onchain actor, and then, if the actor is compliant, route the response to the appropriate, custom implementation of the *Logic* contract.

2.2.1 Requests

Requests are made to the Everest Chainlink node [*] for a boolean indicating whether an onchain address has been linked to the deduplicated identity of a person who has completed KYC.

Requests must contain the address of the onchain actor whose KYC status is being queried and the address of the custom *Logic* implementation contract for callback.

Requests can be made by EOAs or smart contracts.

A fee is taken for each request (see section on *Fees* for more info).

2.2.2 Automation

The Everest Chainlink Consumer contract emits a Fulfilled event when requests are fulfilled. Chainlink's offchain Automation nodes continuously monitor for this event, calling the Router contract with the fulfilled response, which is then routed to the appropriate Logic implementation if the requested address is compliant.

2.2.3 IERC677Receiver

The Router implements the ERC677 Receiver interface to enable requests in a single transaction using the LINK token's `transferAndCall()` functionality [*].

2.2.4 Upgradeability

The Router implements the TransparentUpgradeableProxy [*] to store Chainlink Automation variables such as the Forwarder address and Upkeep ID as immutable. This will save on gas from SLOADs [*].

2.3 Logic

The CLC Logic contract is abstract and must be inherited to implement Contract Level Compliance. Custom implementations of the CLC Logic contract can be made by anyone.

Logic contracts include an internal `_executeLogic()` function that must be overridden, where the code for a regulated activity may be defined. This internal function is called by an external function that can only be called by the Router. When a request is fulfilled and the onchain actor is compliant, this function is passed the requested address.

2.3.1 IERC165

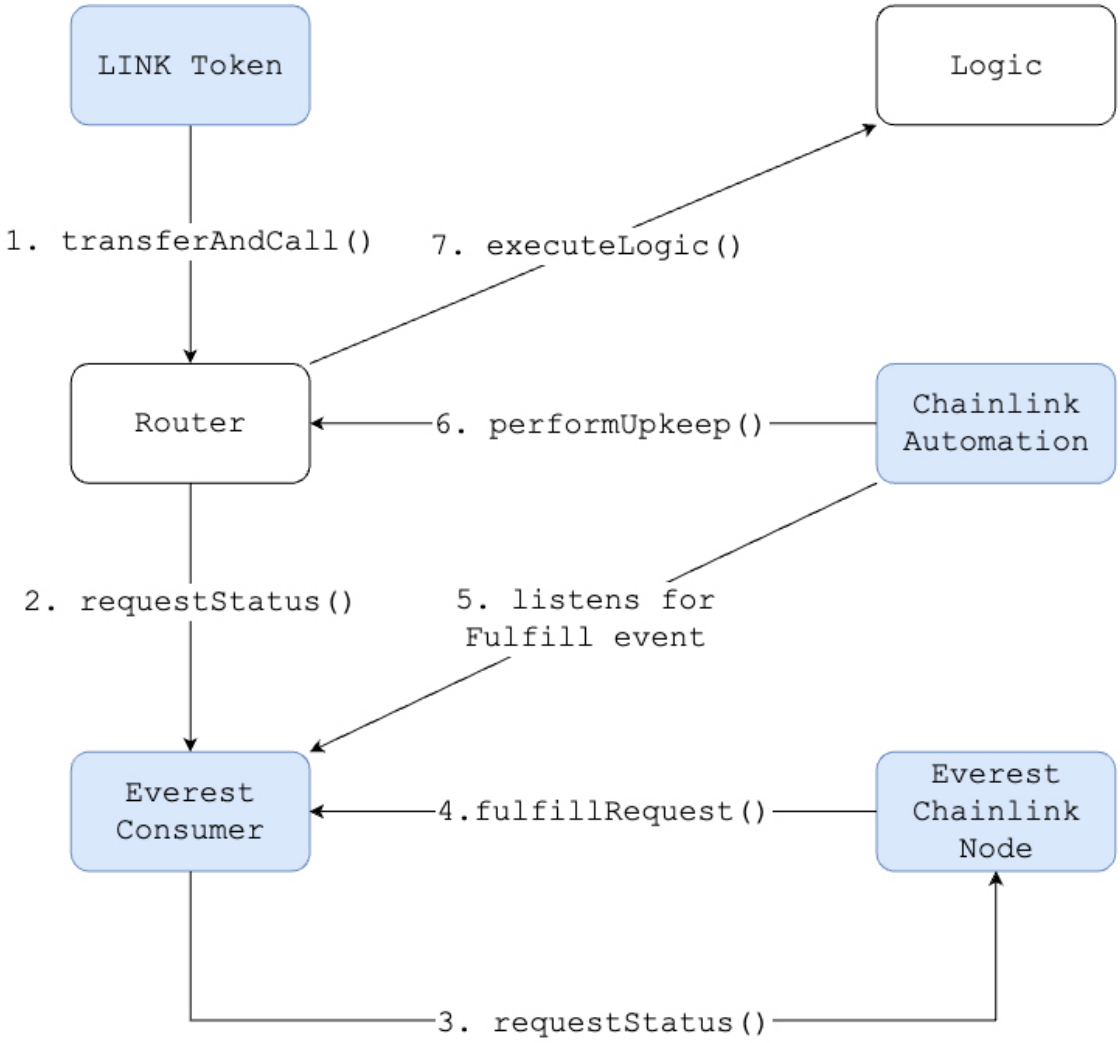
Logic contracts implement the ERC165 standard interface, allowing the Router to check that the Logic address to call does indeed implement the expected Logic interface.

2.4 Calltrace

Requests for the compliant status of onchain actors and a callback passing the fulfilled response to a custom logic contract is the intended functionality for external use, and can be executed in a single transaction.

The CLC Router interacts with a few external services from the *Internet of Contracts* [*] (as well as the CLC Logic contract).

1. **LINK Token** - LINK is used to request the compliant status of an onchain actor. `transferAndCall()` can be used for single-transaction requests, or `requestKycStatus()` can be called from the Router after approving the token.
2. **Router** - The Router updates its state and forwards the request to the EverestConsumer.
3. **EverestConsumer** - The Everest Chainlink Consumer makes a request to the Everest Chainlink node.
4. **Everest Chainlink Node** - The Everest Chainlink node fulfills the request and sends the response back to the EverestConsumer.
5. **Chainlink Automation** - Chainlink offchain Automation nodes monitor EverestConsumer for fulfilled request events. If a pending request exists, the Chainlink Automation Forwarder calls `performUpkeep()` on the Router.
6. **Router** - The Router updates its state and forwards the fulfilled response to the appropriate Logic contract.
7. **Logic** - If the onchain actor has completed KYC, the restricted logic executes on their behalf.



2.5 Crosschain Interoperability

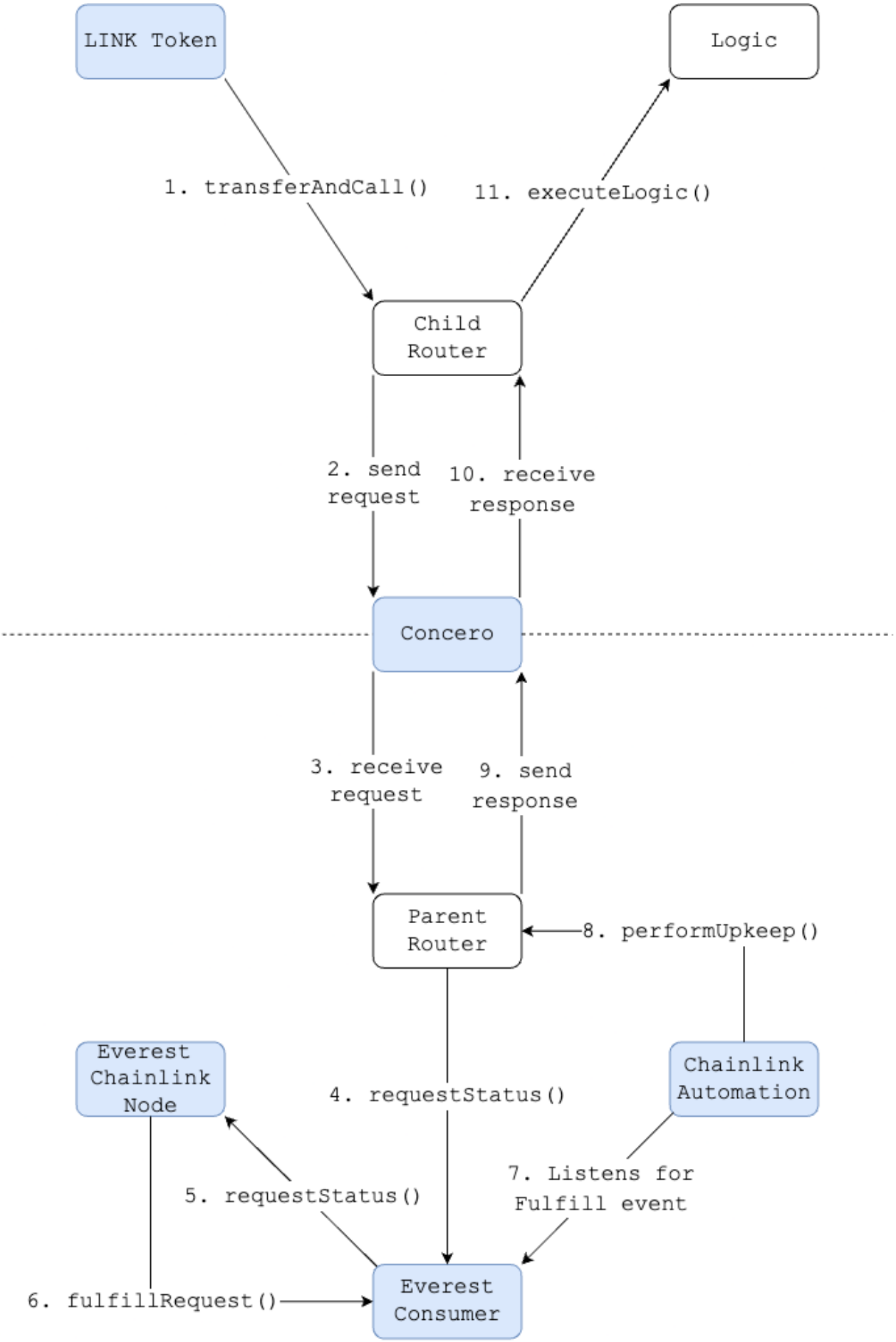
The Everest Chainlink node only needs to provide identity data to a single Concero Compatible Chain for Contract Level Compliance to be available on every CCC.

A “Parent” Router will be deployed on the chain where the Everest identity data is available and any requests made on other chains will be made to a “Child” Router on that chain. The Child Router will interact with the Concero infrastructure to route the request (and fee in LINK) to the Parent Router, which will route the fulfilled response back to the appropriate Child and Logic implementation.

2.5.1 Crosschain Transaction Flow/Calltrace

The transaction flow for a crosschain request and logic callback introduces additional steps of sending and receiving messages between “Child” and “Parent” Routers, through the Concero infrastructure.

LINK will need to be sent from the Child chain to the Parent chain in order to pay for the Automation and Node request services. Alternatively, a payment system allowing users to initiate requests in other tokens that get swapped to LINK on the parent chain could be used.


LINK Token
Logic
1. transferAndCall()
11. executeLogic()
Child Router
2. send request
10. receive response
Concero
3. receive request
9. send response
Parent Router
8. performUpkeep()
Everest Chainlink Node
Chainlink Automation
4. requestStatus()
5. requestStatus()
7. Listens for Fulfill event
6. fulfillRequest()
Everest Consumer

3. Security

The highest priority for Contract Level Compliance is, and always will be, security. Smart contracts must behave as intended, and never behave in any way that is not intended.

3.1 Testing

All Contract Level smart contracts must undergo the following minimum testing process:

- Unit Tests
- Integration Tests
- Static Analysis
- Invariant Tests
- Formal Verification
- Mutation Tests
- Test Networks

3.1.1 Unit Tests

Unit tests cover the most basic functionality of smart contracts and should have 100% code coverage. Every single line of code must be tested. Live mainnets should be forked to make these tests as realistic as possible. Inputs should be fuzzed (randomised) when possible.

Unit tests will be written using Foundry [*], a smart contract development environment that allows tests and scripts to be written in Solidity, the same programming language that is used for the smart contracts.

3.1.2 Integration Tests

Integration tests cover smart contracts functioning correctly within the broader system by testing interactions between the contracts, external dependencies, and deployment processes. These tests will cover Contract Level scripts, including contract deployment and other key interactions, ensuring that they execute as intended. Testing these scripts is essential to confirm that they operate correctly under real-world conditions and integrate seamlessly with the overall system.

3.1.3 Static Analysis

Static analysis is the process of examining smart contracts without executing them to detect common vulnerabilities, logical errors, and optimisation opportunities. It is a necessary step in development because it can help catch issues with relatively little effort.

Leading static analysis tools that can be used include Slither [*] and Aderyn [*].

3.1.4 Invariant Tests

Invariants are properties about a system that must always hold true. Invariants must be clearly defined and tested using a Foundry test suite designed to simulate a realistic environment of fuzzed inputs and external calls. “Thinking in invariants” starts immediately

in the smart contract development process. Writing invariant tests in Foundry allows us to build upon the unit test suite already developed in Foundry.

The foundational invariant of Contract Level Compliance is *restricted logic must only be executed on behalf of actors who have completed KYC with Everest*. Conversely, the polar opposite invariant should also be tested - *restricted logic must never be executed on behalf of actors who haven't completed KYC with Everest*.

In addition to this, appropriate state changes and event emissions should also be tested as invariants, ie *requests emit the correct event*. A request that is pending fulfilment must have paid the correct fee to the correct services, and must not still be pending once it has been fulfilled.

3.1.5 Formal Verification

Formal verification is the process of proving that a smart contract behaves as intended by converting its logic into mathematical formulas and checking them against predefined rules. Unlike traditional testing, which only evaluates a subset of possible contract states, formal verification mathematically proves that the contract's behaviour is always correct under all conditions.

The Certora Prover [*], a leading tool in formal verification, should be used to formally verify invariants. Unit tests should be rewritten as Certora rules [*], transforming predefined test cases into provable conditions. The Certora Verification Language (CVL) [*] enables acutely precise definitions of specified behaviours, allowing for rigorous enforcement of these rules and system invariants.

3.1.6 Mutation Tests

Mutation testing introduces artificial bugs into forked copies of smart contracts in order to improve our previously written invariant tests and formal verification specifications. If introduced bugs in the mutated contracts are not detected by the specification file, this indicates gaps in the verification process. In such cases, the specification must be refined to strengthen its ability to identify unexpected behaviours and enforce the intended contract logic with greater accuracy.

3.1.7 Test Networks

CLC Routers and example Logic implementations will be deployed on test networks for further evaluation of behaviour.

3.2 Audits and Bug Bounty

Before deploying to production, rigorous security reviews must be undertaken. Multiple, independent private reviews from well regarded audit firms must be conducted to identify potential vulnerabilities, assess contract logic for correctness, and ensure adherence to best security practices. These reviews will include manual code reviews and adversarial testing to uncover any weaknesses before the contracts are deployed.

Leading security firms that provide private reviews include Pashov Group [*], Cyfrin [*], Guardian Audits [*], and Certora [*].

The next step is a competitive audit on a platform like Code4rena, CodeHawks, Sherlock or Cantina. Unlike private security reviews, these competitions employ the collective expertise of the wider smart contract security community, including many independent researchers who are enthusiastic for the opportunity to prove themselves professionally. Every additional set of eyes on the code before deployment increases the likelihood of identifying edge-case exploits and critical weaknesses, significantly reducing risk.

It is entirely unacceptable to deploy any smart contract to production that has not been thoroughly audited because the risk of vulnerabilities still being present is greatly increased in unaudited smart contracts.

Once audits are complete, a bug bounty program should be in place for the deployed Router contracts on a platform like Immunefi [*]. Even with rigorous audits and formal verification, no security process can guarantee absolute invulnerability—a bug bounty incentivizes continuous, crowdsourced security testing, allowing ethical hackers to identify vulnerabilities before malicious actors do. By offering rewards for responsibly disclosed exploits, the program provides an ongoing layer of defense, ensuring that any overlooked issues can be fixed before they are exploited in production.

3.3 Gas Limit

Gas limits will be enforced on callbacks to custom logic implementations to mitigate potential gasbombing or denial of service attacks. A default gas limit will be defined as a constant in the Router, with configurable limits between minimum and maximum values allowed. The maximum limit will be within the Chainlink Automation perform limit.

3.4 Multisig

A Safe [*] multisig wallet must be used as an admin to authorise any proxy upgrades to the Router, as well as for Chainlink Automation Upkeep ownership, to prevent a centralised point of failure in the form of a single private key.

The Router contract(s) will have access control for withdrawing accumulated fees. This functionality will, in a later iteration to the infrastructure, be delegated to a *revenue handler* contract. However until that Contract Level infrastructure is securely in place, a multisig must be used here also.

4. Fees

Each request takes a fee, covering the costs of external services provided by Everest, Chainlink and Concero, as well as the Contract Level protocol itself.

Fees are paid in LINK. Payment abstraction allowing requests to be paid in native or alternative tokens could be introduced in a later iteration.

Fee Type	Purpose	Applies To
Everest	Identity data request	All requests
Chainlink Automation	Upkeep execution costs	All requests
Concero	Crosschain transactions	Crosschain requests
Contract Level	Development and maintenance	All requests

4.1 Everest Fee

Covers the cost of the request to the Everest Chainlink node.

4.2 Chainlink Automation Fee

Covers the minimum balance required for the associated Chainlink Automation upkeep to perform.

4.3 Concero Fee

Covers the cost of crosschain interactions facilitated by the Concero infrastructure. This fee is only taken for crosschain requests.

4.4 Contract Level Fee

Covers the costs of ongoing development and maintenance of the Contract Level ecosystem.

5. Future Developments

Once everything that has been laid out so far in this paper has been achieved, there will be ongoing developments to the Contract Level ecosystem.

5.1 Compliant Use Cases

The Contact Level *Compliance* infrastructure has been built from the beginning with modularity and usecase in mind. The Contract Level team will build new smart contracts on top of the CLC system. The exact use cases, and in which order they are built, are yet to be determined, but will likely include compliant insurance contracts and compliant derivatives contracts.

CLC “wrappers” for, or forks of, popular smart contract protocols will also be explored as a usecase.

The universe of traditional institutions and public actors united in their smart contract operations awaits.

5.2 Native Token

There are currently no plans to deploy a native Contract Level token, and its specifics are yet to be determined. However such a token would adhere to the following specifications:

A Contract Level (LEVEL) token will have a fixed max supply across chains. The token will follow the ERC677 and CCT (Cross-Chain Token) standards. The purpose of the token will be Contract Level revenue distribution.

References

[1] Everest Identity Oracle - https://developer.everest.org/#everest-identity-oracle

[2] Chainlink Automation - https://docs.chain.link/chainlink-automation

[3] Concero V2 - https://blog.okron.cc/posts/concero-v2-technical-overview

[4] FSB Global Regulatory Framework for Crypto-asset Activities - https://www.fsb.org/2023/07/fsb-global-regulatory-framework-for-crypto-asset-activities

[5] MiCA - https://www.esma.europa.eu/esmas-activities/digital-finance-and-innovation/markets-crypto-assets-regulation-mica

[6] US Presidential Executive Order - https://www.whitehouse.gov/presidential-actions/2025/01/strengthening-american-leadership-in-digital-financial-technology

[7] Identification requirement in traditional activity - https://www.trulioo.com/industries/crypto-identity-verification/kyc

- https://www.fca.org.uk/firms/financial-crime/money-laundering-regulations

[8] Centralised exchanges required to adhere to KYC regulations - https://notabene.id/crypto-travel-rule-101/kyc-crypto

[9] Onchain identification requirements - https://public-inspection.federalregister.gov/2024-30496.pdf

- https://www.forbes.com/sites/shehanchandrasekera/2024/12/27/understanding-the-new-irs-defi-broker-tax-regulations
- https://blockworks.co/news/bipartisan-bill-defi-project-controllers

[10] Requirements for regulated entities facilitating KYC - https://www.fca.org.uk/firms/financial-crime/money-laundering-regulations

[11] Decentralisation and self-custody in smart contracts and blockchain - https://ethereum.org/content/whitepaper/whitepaper-pdf/Ethereum_Whitepaper_-_Buterin_2014.pdf

- https://bitcoin.org/bitcoin.pdf

[12] Chainlink as an industry standard - https://www.swift.com/news-events/press-releases/swift-ubs-asset-management-and-chainlink-successfully-complete-innovative-pilot-bridge-tokenized-assets-existing-payment-systems

[13] Everest as only provider of sybil resistant identity data - https://everestdotorg.medium.com/everest-is-now-a-data-provider-on-the-chainlink-network-bringing-novel-identity-data-to-377039492189