TLDR

L2 neworks such as rollups, validiums and optimiums primarily use execution proofs (ZK proofs or fraud proofs) to prove (or disprove) transaction and state transition validity to the base layer (Ethereum). While proofs of execution are important, they do not detect many categories of faults and frauds that can happen in the various components that comprise the overall layer-2 infrastructure. Such faults/frauds are difficult to prove and impossible to attribute and penalise on-chain.

This post introduces integrity proofs, which complements zk proofs and fraud proofs for rollup security. Integrity proof is a mechanism to attest to the integrity of services running in a particular instance of rollup, providing assurance to the end users that no malicious or other unauthorized code changes have been performed in the production environment. Integrity proofs use TEEs to deploy and monitor rollup services and report any breach of service integrity on a new layer-2 network (let's call it Integrity verification chain) that provides decentralised security services to rollups.

Layering integrity proofs over existing execution proofs greatly improves the censorship resistance of the rollup, detects unauthorised MEV extraction and offers protection from other types of malicious activities in the rollup infrastructure from privileged or unauthorized users. This design also reduces the trust assumptions , compared to the current architecture of centralised rollup deployments that rely disproportionately on execution proofs to demonstrate offchain execution validity.

Introduction

Layer-2 networks settling on Ethereum (aka base layer) predominantly use either fraud proofs (proofs of invalidity) or ZK-proofs (proofs of validity) to prove transaction validity and updated state commitments to the base layer. Both these are execution proofs that cryptographically attest to those activities that are objectively recorded and verifiable on-chain (aka running in the EVM or equivalent VM).

However, such execution proofs alone are inadequate to verify the security of the overall layer-2 network. This is because there are many centralised components in L2s where faults and malicious activities can occur, which cannot be proven on-chain (non-attributable faults). There are many solutions being worked on to address centralisation risk in rollups, but they come with their own tradeoffs including loss of sovereignty/control, reducing utility of own token, cost increase or increased complexity of architecture.

This post proposes a solution to detect unattributable faults

while running rollup services (of the kind that cannot be detected by ZK proofs alone) , without any loss of sovereignty or loss of control for the project deploying the layer-2 network. (Note: The terms layer-2 network and rollup are used interchangeably in this post as a broad term to encompass rollups, validiums and optimums.)

Integrity proofs are proposed as a mechanism to certify to the presence (or absence) of unattributable faults in the operation of a rollup service, by having an observer running in a Trusted Execution Environment (TEE) provide cryptographic attestation of its own execution environment and signing the attestation with its own in-enclave generated cryptographic identity (signing key) that cannot be tampered with even by the operator of the attestation service.

Problem space

Layer-2 networks (both optimistic and Zk stack chains) have many components such as rpc nodes, sequencers, pool & state databases, executors, provers, L1 transaction managers etc that are basically deployed either as platform-native binaries or docker images. Examples of non-attributable faults/frauds include censoring transactions based on addresses or geo-fencing access in json-rpc node, changing the ordering algorithm of sequencer, making unauthorised/undetected changes to the state or pool database, and withholding transaction batch data posted to external DA layer. Currently, users make heavy trust assumptions while using L2 chains that such malicious or unauthorised actions don't take place in the centralised services operated by their provider. More specifically, users make the following trust assumptions:

a) they trust the rollup project foundations (eg Polygon, ZK, Optimism, Arbitrum) to not introduce any malicious code on the published rollup service images.

b) trust that the right security controls are in place such that employees/contractors that belong to the rollup project or RaaS operators have restricted privileged access (or that those who do , do not abuse privileged access)

In short, users rely on the security offered by the reputational credibility (social & commercial) of these trusted third parties. Integrity of the rollup services operated by these parties is not verifiable, and any faults are not attributable or provable on-chain. This goes against the basic premise of blockchain networks aka verifiability.

Solution proposed

[

Integrity Verified Services

1920×1429 112 KB

](https://ethresear.ch/uploads/default/original/3X/e/d/edb7cc61ad8bf9c863c1a9d1deeef28720e7a109.jpeg)

Figure shows the overview of the proposed solution, which has a new layer-2 network acting as the integrity verification chain (let's call it IVC in short), and three types of actors participating in it - rollup framework providers, rollup projects and TEE operators. Let's understand this better using the example of a single rollup service- the sequencer (which needs no introduction).

1. Rollup framework providers (eg polygon cdk, Zkstack, OP stack , Arbitrum etc) register the service image of their latest release of sequencer with IVC.

2. A new rollup project comes along and wants to deploy the sequencer from one of these frameworks. The request is sent to a Host program

(creatively named) that runs in a TEE. The host program retrieves the service image registered by the provider, and deploys it on a VM that's specified by the rollup provider. (The only requirement is for the host program to be given SSH access to deploy the service in the VM). For simplicity, let's assume this service image is a docker image. The host program downloads the docker image and instantiates a new container using this image. The container Id is captured by the host program and recorded on IVC smart contract designated for the purpose.

1. Periodically (based on frequency specified by the rollup project owner) the host program verifies if the container id of the running sequencer matches that recorded on the IVC layer-2 smart contract. If there is any discrepancy, it flags it as a violation of integrity on the integrity verification chain (IVC).

2. The host program can generate an integrity proof on demand (or it can be generated unconditionally in the protocol workflow) attesting to the integrity of a particular service that is deployed and monitored by it. The integrity proof would contain the details of the rollup service that's being monitored, proof of the execution environment in which the host program runs including the quote/remote attestation parameters and its own binary fingerprint (MRENCLAVE), Further, the integrity proof generated by the host program can be signed by a keypair that's generated within the enclave so that the operator of the host program VM cannot make a false attestation. The signing key would not be accessible even to the operator.

Here, the sequencer component that was used as the example, becomes the Integrity-verified service (IVS). Technically, the sequencer itself can be run inside a TEE, but that would be another solution not discussed here.

How does integrity verified service offer a better security model for rollups?

In this example, let's assume that after the rollup service is deployed , someone with privileged or unauthorised access modifies the sequencer code to add censoring logic or ordering algorithm, and restarts the docker container. This will result in a change to the container finggerprint, and this will immediately be detected by the host program (as the running container id will not match the registered container id that was started by the host program). The integrity proof from the host program can certify to the presence or absence of such non-attributable faults.

When integrity proofs are combined with prevailing proof mechanisms such as zk validity proofs or fraud proofs, it strictly improves the status quo in terms of rollup security.

Benefits of this solution

For end-user:

The end user of the rollup/L2 chain is now assured that in addition to their on-chain transactions being secure (due to natively built-in zk proofs or fraud proofs), additionally any non-attributable faults that can occur in sequencer such as censoring or ordering algorithm changes will immediately be detected and flagged for attention. The user can verify integrity proofs in a permission-less manner.

For rollup project owner:

They don't just have to trust their own devops teams or that of the RaaS operator that they outsource to, but they can verify the integrity of the rollup services deployed permissionlessly, on an ongoing basis. They get a higher security layer-2 chain (trust-minimised) compared to what they would get by deploying a standard rollup stack out of the box.

For RaaS operators

: They have lower operational risk by ensuring that the right service images are deployed everytime. This also results in lower risk of reputational/business loss due to unauthorised access or security attacks on their infra.

For rollup framework providers

: By publishing their service images and actively participating in the integrity verification activity, they attract better quality of builders and projects on their ecosystem.

Further security enhancements

The integrity proofs described solve two attack vectors on rollup services:

1. Any unauthorised changes to the deployed rollup service is detected and flagged.
2. The correct secure version of each rollup service is deployed in the production environment

This is a clear improvement from having just execution proofs. But there can be other types of attacks that need to be designed for. One example is that someone with privileged access can still log into the running docker container and change environment variables. To address this, the host program can be enhanced to periodically log into the running containers and compare the environment parameters on the current running version of container vs the original deployed version.

There can be other types of attacks that are more difficult to detect. for example a sophisticated actor can log into the docker container and manipulate memory . Memory encryption techniques can be evaluated to protect against such attacks. Other solutions could involve running multiple instances of a rollup service (eg sequencer or prover) governed by a consensus algorithm, but it increases costs of operating the rollup infrastructure, and it will be the decision of the rollup project to evaluate the level of risk they are willing to accept and the cost they are willing to pay.

Summary

The premise of this solution is that a combination of existing execution proof techniques

(zk or fraud proofs) for objectively-verifiable faults in combination with Tee-based integrity proofs

for non-attributable faults offers a superior trust-minimised security model for rollups, compared to the status quo.