RollGuard: Defending RPC Manipulation Attacks in Optimistic Rollups with Graph ML

Abstract—**The rapid adoption of Layer 2 (L2) blockchain scaling solutions, such as optimistic rollup (OR), have introduced new vulnerabilities that compromise the security and efficiency of blockchain. In this work, we introduce a critical security vulnerability in the OR system, where attackers exploit remote procedure call (RPC) address manipulation to deceive verifiers into challenging legitimate state roots. By altering the sender's RPC address during transaction processing delays, attackers create a false perception of fraud, resulting in unwarranted challenges and penalties for verifiers, thereby undermining the protocol's integrity. To mitigate this flaw, we propose RollGuard, a graph machine learning (ML) approach that dynamically models interactions within the blockchain as a graph, with nodes representing senders and receivers (RPC endpoints). Utilizing a graph neural network (GNN), this method detects unusual RPC address changes by analyzing real-time and historical transaction patterns, enabling proactive identification of potential attacks. The GNN flags anomalous behavior in RPC address updates, offering early alerts to verifiers and reducing false challenges. Our experimental results show that RollGuard framework significantly enhances protocol security, lowering false-positive challenge rates and protecting verifiers from undue penalties.**

Index Terms—**Blockchain, optimistic rollup, graph neural network, machine learning**

I. Introduction

Blockchain technology has seen widespread adoption due to its decentralized, secure, and transparent nature. However, as blockchain networks grow, they encounter scalability challenges, leading to slower transaction speeds and higher costs [1]. To address these limitations, Layer 2 (L2) scaling solutions, such as optimistic rollups (ORs), have been developed to increase throughput while maintaining security [2]. ORs achieve scalability by processing transactions off the main chain and periodically submitting a condensed state root to the Layer 1 (L1) blockchain for validation [3]. While this approach enhances transaction efficiency, it also introduces new vulnerabilities that can be exploited by attackers to compromise network reliability.

OR protocols depend on the remote procedure call (RPC) endpoints [4] to relay transaction data to aggregators. Prior research has identified security risks associated with RPC endpoints, such as unauthorized access and manipulated responses. Cheng et al. [5] demonstrated how unprotected JSON-RPC endpoints in Ethereum nodes can be exploited by attackers to initiate unauthorized Ether and ERC-20 token transfers by sending maliciously crafted RPC requests. Additionally, a study by Chainstack [6] highlighted the exposure of RPC endpoints in decentralized applications (DApps), emphasizing the risk of unauthorized access and data breaches due to inadequate security measures. Furthermore, Li et al. examined how vulnerabilities in RPC services could be leveraged to disrupt decentralized applications [7], leading to denial-of-service attacks and other malicious activities. However, a more sophisticated attack vector—*RPC address manipulation*—remains unexplored. In this paper, we identify and examine a critical vulnerability in the OR protocol, where attackers can exploit the RPC address feature to manipulate transaction verification.

Specifically, this attack leverages the delay in transaction processing between the validation of a transaction batch by aggregators and the submission of the corresponding state root to verifiers on the L1 chain for verification. During this delay, an attacker, being the sender of one or more transactions, alters its RPC address, creating a discrepancy that misleads verifiers into challenging a legitimate state root. Such challenges, though incorrect, lead to the slashing of verifier stakes, effectively penalizing them while reducing the system's throughput incurred due to dispute phases. This RPC manipulation exploit not only results in unnecessary verifier losses but also undermines confidence in the OR protocol by exploiting a fundamental flaw in its design.

To address this vulnerability, we propose RollGuard, a defense mechanism that utilizes graph machine learning (ML) to detect abnormal patterns in RPC address changes. The RollGuard framework models OR interactions as a dynamic graph, where nodes represent senders and receivers (RPC endpoints) and edges represents the RPC update relation history. RollGuard leverages a graph neural network (GNN) [8] to analyze both real-time and historical RPC endpoint update patterns, effectively identifying irregular changes in RPC addresses that indicate potential manipulation. The GNN flags transactions with unusual patterns for further review before challenges are initiated. This graph ML-based approach enables proactive detection, reducing false-positive challenges and minimizing the risk of unwarranted penalties for verifiers. Our contributions are three-fold:

- We identify a novel security vulnerability in ORs, where attackers exploit RPC address manipulation to deceive verifiers and impact the L2 throughput.
- We develop and validate RollGuard, a graph ML-based framework to defend against this attack, enhancing the security and resilience of the OR protocol.
- We conduct extensive experimental evaluations demonstrating that RollGuard significantly reduces false-positive challenge rates, safeguarding verifiers from undue penalties while maintaining high protocol throughput.

Section II provides the necessary background information,

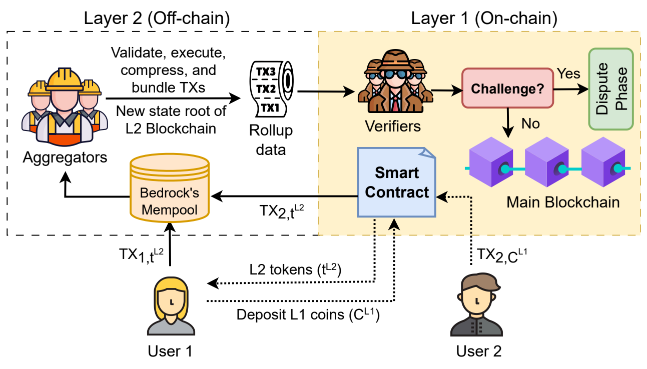


Fig. 1. Optimistic Rollup workflow.

while related works are reviewed in Section III. The RPC manipulation attack is introduced in Section IV. Section V presents the technical details of the proposed defense, ROLLGUARD. Empirical analysis and findings are discussed in Section VI. Finally, the paper is concluded in Section VII.

II. BACKGROUND

In this section, we present some preliminary concepts that will help explain the attack and corresponding defense.

A. Rollups

Blockchain rollup is a scaling solution that addresses the scalability limitations of traditional blockchain networks by bundling multiple transactions or smart contracts together before processing them on the main blockchain [2]. It aims to enhance the throughput and efficiency of blockchain systems, especially those utilizing smart contracts and decentralized applications. Rollups work by aggregating transactions off-chain, computing their outcomes, and submitting a summary or proof of those outcomes to the main blockchain. This significantly reduces the computational load and congestion on the main chain while maintaining the security guarantees through cryptographic techniques. There are two main types of rollups: ZK-Rollups [9], which utilize zero-knowledge proofs to validate the correctness of transaction batches, and ORs, which rely on a challenge and dispute mechanism to ensure the accuracy of off-chain computations.

B. Merkle Tree Root

In the context of blockchain rollup, a Merkle root plays a crucial role in ensuring the integrity and security of off-chain transactions or smart contracts before they are submitted to the main blockchain [10]. The Merkle root is a cryptographic hash that represents a summary of all the individual transactions or data within a specific batch. By combining these transaction hashes using a Merkle tree structure, the Merkle root condenses a large amount of information into a single value. This Merkle root is then included in the transaction submitted to the main blockchain, serving as a proof of the correctness of the off-chain computations. In case of any dispute or challenge, the full data can be verified on-chain by recreating the Merkle tree and comparing it to the provided Merkle root, ensuring the accuracy of the summarized data.

C. Workflow of Optimistic Rollup

OR addresses scalability concerns of blockchain by processing transactions off-chain and ensuring their validity through an optimistic approach, backed by a challenge mechanism. The workflow of OR, illustrated in Figure 1, involves several sequential steps. First, users must acquire L2 tokens (t^{L2}) to interact with the rollup system, which can be exchanged for other cryptocurrencies (C^{L1}) via the L1 smart contract. L2 transactions are then submitted to Bedrock's Mempool [11], where aggregators collect and process them. Users can submit their transactions directly to the Mempool or route them through the L1 smart contract. Aggregators process these transactions, compute cryptographic aggregates, generate the Merkle state root for the L2 chain, and send the results to verifiers. Verifiers on L1 monitor these submissions, identifying and disputing fraudulent or invalid transactions within each batch. If fraud is suspected, a challenge period is initiated, allowing verifiers to present fraud-proofs to contest the optimistic assumption (Figure 2). If the fraud is confirmed, the disputed transactions are reverted, and the malicious party loses their security deposit. On the other hand, if no valid challenge is made within the dispute window, the transactions are finalized and incorporated into the blockchain [9].

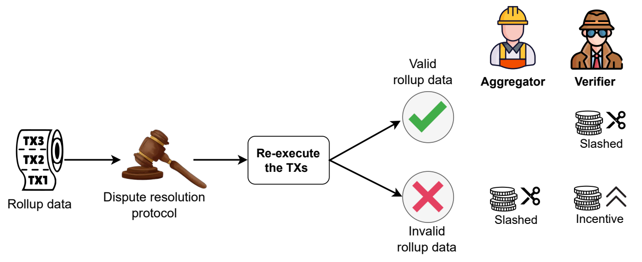


Fig. 2. Dispute Phase workflow.

D. Graph Neural Network

GNNs are a class of deep learning models designed to process data structured as graphs [8]. Unlike traditional neural networks that operate on Euclidean data (like images or sequences), GNNs handle non-Euclidean structures, such as social networks, molecular structures, or transportation grids. By iteratively aggregating and transforming information from a node's neighbors, GNNs learn representations that capture both local and global graph properties [12]. This enables them to perform tasks such as node classification, link prediction, and graph classification.

III. RELATED WORK

Vulnerabilities in RPC mechanisms have emerged as a critical threat vector in blockchain, allowing attackers to manipulate transaction processing. SlowMist [13] reported an Ethereum-based scam where users were deceived into modifying their RPC endpoints, leading to falsified account balances and unauthorized transaction approvals. Furthermore, unprotected JSON-RPC endpoints have been identified as a security risk in Ethereum nodes. Research by Cheng et al. [5] demonstrated how attackers could exploit these endpoints to initiate unauthorized Ether and ERC-20 token transfers by sending maliciously crafted RPC requests. Given these vulnerabilities, recent research has explored the use of GNNs for anomaly detection in blockchain systems. Chang et al. [14] introduced a GNN-based approach for detecting anomalous blockchain nodes, effectively identifying fraudulent activities through graph-based learning. Similarly, Sharma et al. [15] proposed a self-supervised GNN model leveraging deep graph infomax (DGI) to enhance fraud detection in decentralized ecosystems. Unlike previously reported RPC-based attacks, which primarily focus on direct endpoint manipulation or unauthorized access, our proposed RPC URL manipulation attack introduces a novel adversarial strategy where attackers dynamically alter RPC addresses during transaction processing delays to deceive verifiers into issuing false challenges. This subtle yet effective manipulation creates a false perception of fraud, leading to unnecessary disputes and penalties, ultimately undermining the integrity of ORs. Existing GNN-based defenses primarily focus on static anomaly detection or heuristic-based filtering. However, our proposed defense mechanism, RollGuard, differs by dynamically modeling blockchain interactions and detecting suspicious RPC address transitions in real-time. By incorporating both historical transaction patterns and real-time behavioral analysis, RollGuard provides a proactive defense against adaptive adversaries.

IV. RPC Manipulation Attack: Key Idea

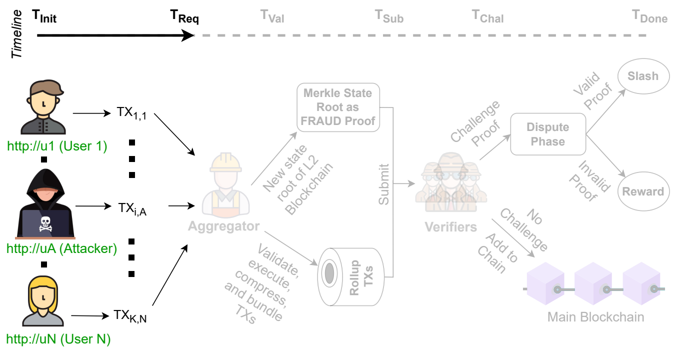
This section introduces the identified RPC manipulation attack technique, leveraging which a user can deceive the verifier to initiate an incorrect dispute phase. This attack occurs over a sequence of stages, each representing a distinct phase in the timeline of the attack. Figure 3 illustrates the progression of the attack, with each subfigure corresponding to a specific stage. The following subsections detail each stage.

A. Transaction Submission

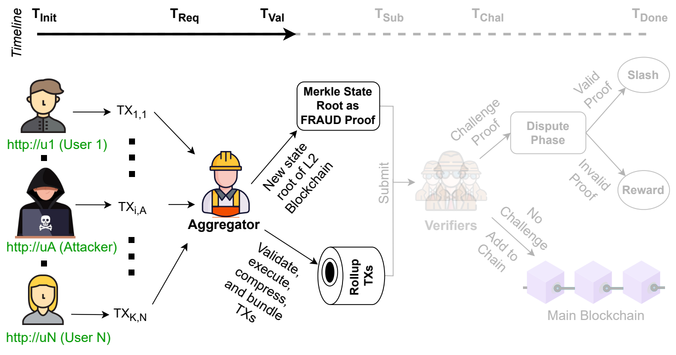
In the initial stage (Figure 3(a)), the OR system operates as intended. Users interact with the system by submitting their transactions through designated RPC endpoints. For example, User 1 uses the endpoint `http://u1` to submit its first transaction $TX_{1,1}$, while User N submits its k-th transaction $TX_{K,N}$ through `http://uN`. The adversarial user (i.e., attacker) uses its current endpoint `http://uA` to submit its i-th $TX_{i,A}$ transaction.

B. Transaction Validation and Merkle Root Computation

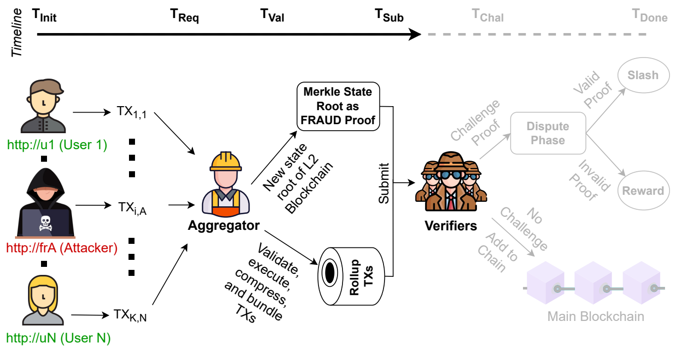
In the second stage, the *Aggregator* validates the submitted transactions before aggregating them into a batch (Figure 3(b)). It first executes the transactions and compiles them into rolled-up batches. Subsequently, it calculates the Merkle root of the updated L2 chain based on the newly processed transaction data. The rolled-up transaction batch and



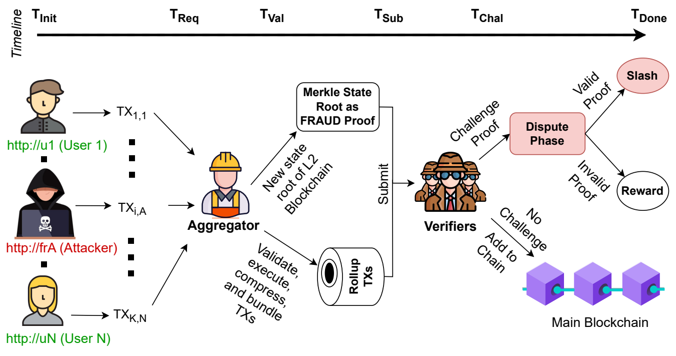
(a)



(b)



(c)



(d)

Fig. 3. The consecutive phases of the RPC URL manipulation attack: (a) transaction submission, (b) transaction validation and proof computation, (c) attack initiation, and (d) verifier deception.

the corresponding computed state root are then submitted for verification in the L1. At this point, the system operates as

TABLE I
LIST OF NOTATIONS

Symbol	*Definition*
α	Probability of attack in terms of malicious user percentage
τ	Risk threshold: confidence level needed for attacker classification
λ	Probabilistic challenge rate
$\mathcal{B}$	Batch of transactions in consideration
$\mathcal{G}^t$	Graph formed at t-th training instance
S_{RPC}	Sender's RPC URL address
R_{RPC}	Receiver's RPC URL address
S_{URLs}	Count of RPC URL updates for user 'S'
F_n	Features of nodes in graph
F_e	Features of edges in graph
$\mathcal{E}$	DNN Model embeddings
m_n	Message of node n

expected, with the adversarial user continuing to utilize its original RPC endpoint (i.e., `http://uA`).

C. Attack Setup

In the third stage, the attacker launches the RPC manipulation attack by using an updated fraudulant RPC endpoint (`http://frA`). As shown in Figure 3(c), the attacker exploits delays between the batch validation (T_{Val}) by the *Aggregator* and the batch submission (T_{Sub}) to the *Verifier*. Specifically, the attacker dynamically modifies the RPC endpoint to fabricate a fraudulent Merkle root representation for the rollup batch, which differs from the actual transaction data. This manipulated state root creates a deceptive appearance of fraud during the verifier's validation process. By strategically delaying the RPC manipulation, the attacker ensures that the Aggregator computes the correct state root, while the Verifier later detects an inconsistency between the computed state root and the actual transactions due to the altered sender address (i.e., the attacker's modified RPC entry).

D. Verifier Deception and Attack Consequences

During this stage, the *Verifiers* are misled into interpreting a legitimate state root as fraudulent (Figure 3(d)). Relying on the altered data from the attacker's manipulated RPC endpoint, the *Verifiers* perceive a discrepancy between the submitted state root and the actual transaction state. This false perception prompts them to mistakenly flag the batch as fraudulent. As a consequence, *Verifiers* initiate fraud challenges during the Dispute Phase, under the assumption that the system has been compromised. These unnecessary challenges result in disruptive dispute, increasing overhead and reducing protocol efficiency. *Verifiers* who submitted false challenges face penalties (referred to as slashing), as their challenges are deemed invalid during the dispute resolution process.

V. TECHNICAL DETAILS OF ROLLGUARD

In this section, we present the technical details of ROLLGUARD, the proposed GNN-based framework to defend against the RPC URL manipulation attack. Table I presents all the notations used in explaining the model.

A. Integration of ROLLGUARD in Optimistic Rollup System

Here, we describe how the proposed ROLLGUARD framework integrates into the OR workflow to mitigate RPC manipulation attacks. During transaction processing, ROLLGUARD continuously monitors interactions (not shown in the Figure), dynamically modeling OR transactions as a graph. This allows the framework to detect anomalies in RPC address changes, which are characteristic of manipulation attempts. By analyzing both historical and real-time transaction patterns, ROLLGUARD proactively identifies this suspicious behavior, ensuring that genuine transactions are not disrupted while minimizing false challenges.

As shown in Figure 4, ROLLGUARD intervenes at a critical stage of this workflow. When verifiers have the suspicion that a batch might have some wrong transaction, they leverage the ROLLGUARD module before challenging the batch. Then, if the wrong information is only related to the mismatch of sender RPC URL and ROLLGUARD detects that the initial user in question is identified as an 'RPC URL manipulator user,' then the batch is deemed valid and added to the main chain. This suspicion management step helps to prevent unwarranted challenges from disrupting the workflow and protects verifiers from penalties caused by false fraud proofs. It also reduces the occurrence of false challenges, preserving the trustworthiness and efficiency of the protocol. In cases of actual tampering of transactions, ROLLGUARD enables the challenging and forwards the batch to the dispute phase.

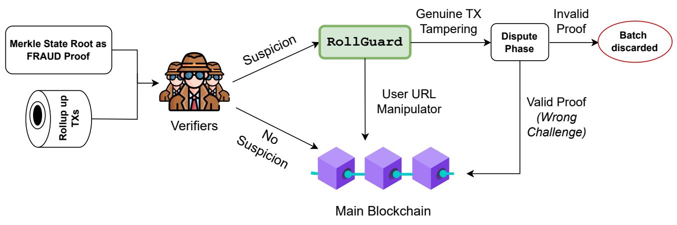


Fig. 4. Proposed ROLLGUARD technique in the optimistic rollup workflow.

B. Regulating the Security-Throughput Trade-off

With ROLLGUARD, verifiers are designed to bypass challenging batches suspected to involve an RPC URL manipulator. However, this mechanism introduces a potential downside: in some instances, verifiers may fail to challenge rollup batches that genuinely contain tampered data due to their association with suspected manipulators. As a result, the compromised data might be added to the L1 chain, undermining its integrity. While ROLLGUARD enhances the system's throughput by avoiding delays caused by unnecessary disputes, this improvement comes at the cost of occasional lapses in security. Consequently, the framework introduces an inherent trade-off between maximizing throughput and ensuring robust security. Thus, the efficiency of ROLLGUARD in mitigating RPC manipulation attacks depends on its ability to balance security and throughput. While ignoring certain challenges improves system efficiency, excessive avoidance can compromise integrity. To regulate this trade-off, we introduce two key parameters: *Risk Threshold* (τ) and *Challenge Rate* (λ), each of which influences the decision-making process.

1) Risk Threshold: The risk threshold (τ) defines the confidence level required for a verifier to classify an entity as an RPC manipulator and ignore challenges against its batches. A higher τ results in more ignored challenges, increasing throughput; however, reducing security since legitimate fraud may go undetected. Conversely, a lower τ enforces a stricter challenge policy, improving fraud detection, however, at the cost of reduced throughput due to more frequent dispute phases. Through experiments (presented in Section VI), we found that an optimal threshold τ^* exists where both security and throughput are maximized without excessive compromise.

2) Challenge Rate: Instead of a binary decision to challenge or ignore transactions, ROLL-GUARD uses a probabilistic challenge rate (λ) to balance false negatives and throughput. A higher λ (e.g., 0.9) means verifiers challenge more suspected manipulations, improving security but reducing efficiency due to increased dispute resolution overhead. Conversely, a lower λ (e.g., 0.2) leads to fewer challenges, enhancing throughput but increasing the risk of tampered batches being finalized. We experiment in Section VI to illustrate how varying λ affects the false negative rate and transaction throughput, highlighting an optimal challenge rate λ^* that achieves a balance between security and efficiency.

C. *Proposed* ROLLGUARD *Algorithm*

The ROLLGUARD algorithm introduces a graph-based framework to detect and mitigate RPC manipulation attacks in blockchain systems. It dynamically models the interaction between nodes (representing RPC endpoints) and transactions, leveraging GNNs for anomaly detection. The Algorithm 1 has been described step-by-step as follows.

1) Graph Construction and Initialization: The algorithm begins by constructing and updating a directed graph $\mathcal{G}^t$ for the current training instance t, using the batch of transactions $\mathcal{B}$. Each transaction Tx is processed to identify the sender RPC ($Tx.S_{RPC}$) and receiver RPC ($Tx.R_{RPC}$). If these nodes do not exist in the graph, they are added. A directed edge is created between the sender and receiver nodes, weighted by the number of URLs associated with the sender ($|Tx.S_{URLs}|$). This step dynamically incorporates new transactions and updates the graph structure, ensuring that $\mathcal{G}^t$ reflects real-time interactions.

2) Feature Extraction: After constructing the graph, node features (F_n) and edge features (F_e) are extracted. These features encode relevant information about the RPC nodes and their interactions, such as transaction frequency, historical behavior, and edge weight trends. This feature extraction process prepares the data for input into the GNN model.

3) Graph Neural Network Initialization: A GNN model is initialized using the updated graph $\mathcal{G}^t$. This model is tasked with learning embeddings for the nodes and edges by aggregating and updating feature representations. These embeddings capture both local and global patterns within the graph, essential for identifying abnormal behavior.

Algorithm 1: Proposed ROLLGUARD Algorithm

```
Function ROLLGUARD ($\mathcal{B}$,$\mathcal{G}^t$,$\tau$,$\lambda$) :
    for $Tx \in \mathcal{B}$ do
        if $Tx.S_{RPC} \notin \mathcal{G}^t$ then
            Add $Tx.S_{RPC}$ as node to $\mathcal{G}^t$;
        end
        if $Tx.R_{RPC} \notin \mathcal{G}^t$ then
            Add $Tx.R_{RPC}$ as node to $\mathcal{G}^t$;
        end
        Add directed edge $(Tx.S_{RPC}, Tx.R_{RPC})$ to $\mathcal{G}^t$
          with weight $| Tx.S_{URLs} |$;
    end
    $F_n \leftarrow ExtractNodeFeats(\mathcal{G}^t)$;
    $F_e \leftarrow ExtractNodeFeats(\mathcal{G}^t)$;
    $Model \leftarrow InitializeGNN(\mathcal{G}^t)$;
    $\mathcal{E} \leftarrow Model.ComputeEmbeddings(F_n, F_e)$;
    for $n \in \mathcal{E}.nodes$ do
        for $i \in Iterations$ do
            $m_n^{(i)} \leftarrow \sum_{u \in Neighb(n)} Aggreg(F_u^{(i-1)}, F_{(u,n)})$
            $F_n^{(i)} \leftarrow Activation(Update(m_n^{(i)}, F_n^{(1-i)}))$
        end
    end
    for $Tx \in \mathcal{B}$ do
        $Score \leftarrow CalculateAnomaly(Tx.S, \mathcal{E}, \tau, \lambda)$;
        if $Score \Leftrightarrow RPC^{Manipulate}$ then
            $Challenge(\mathcal{B})$;
            $Break$;
        end
    end
    $\mathcal{G}^{t+1} \leftarrow update\_graph(\mathcal{G}^t, \mathcal{B})$;
return $\mathcal{G}^{t+1}$
```

4) Anomaly Detection Using Node Embeddings: The GNN computes embeddings $\mathcal{E}$ for all nodes and edges in the graph through iterative message-passing operations. For each node n, embeddings are updated iteratively based on the aggregated features of its neighbors. The iterative process uses an aggregation function to combine information from neighboring nodes and an update function to refine the node's feature representation. An activation function ensures non-linearity, enabling the model to capture complex patterns. These embeddings represent the structural and transactional characteristics of each node and its interactions.

5) Anomaly Scoring and Alerts: Each transaction in the batch $\mathcal{B}$ is analyzed to calculate an anomaly score based on the sender's embedding, using thresholds τ and λ to determine suspicious behavior. If a transaction is flagged as involving an RPC manipulator, an alert is issued, and the associated batch $\mathcal{B}$ is challenged. This proactive detection prevents tampered data from being added to the blockchain.

6) Graph Update: Finally, the graph is updated to $\mathcal{G}^{t+1}$ by incorporating new transactions from the batch $\mathcal{B}$. This step ensures the model remains adaptive to evolving interaction patterns, enhancing its ability to detect emerging threats.

D. *Data Collection for GNN Training*

To train the GNN model, we collected data from publicly available sources that provide comprehensive information

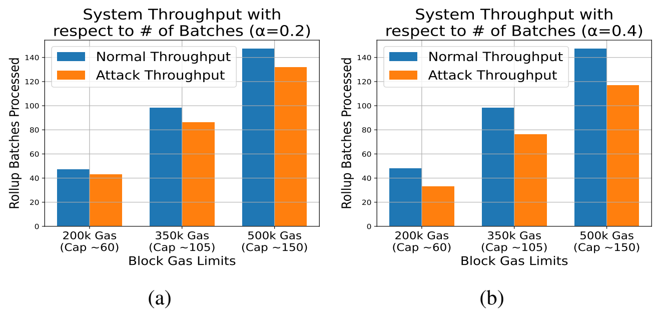


Fig. 5. Impact of the RPC URL manipulation attack on the throughput of the optimistic rollup system, in terms of the number of rollup batches processed with different block gas limit, with (a) attack probability of 20% (α = 0.2) and (b) attack probability of 40% (α = 0.4).

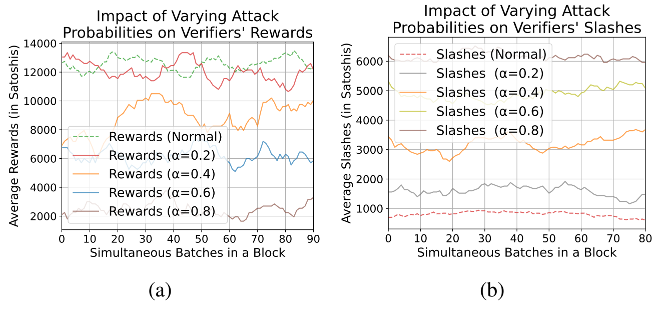


Fig. 6. Effect of varying attack probabilities (α) on the verifiers': (a) average reward gained over the batches of a block and (b) average slashing received over the batches of a block (in Satoshis), and comparison with the benign case without any attack.

about transaction and node interactions. We specifically use snapshots of Optimism, a state-of-the-art OR system. The following datasets were utilized:

- **Optimism Node Snapshots:** The transaction and node interaction data were extracted from the snapshots provided by Optimism's official documentation for node operators [16].
- **Archived Optimism Snapshots:** Additional data were retrieved from an archived repository of Optimism snapshots hosted on the Internet Archive [17].

This data formed the basis for training the GNN model within the RollGuard module.

VI. Evaluation

In this section, we analyze the attack impact on the network throughput and the verifiers' reward and slashing. Later, we validated the effectiveness of the proposed RollGuard technique in defending against the attack.

A. Impact of RPC Manipulation Attack on Throughput

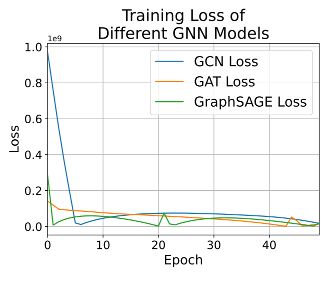
In this section, we analyze the system throughput under normal and attack scenarios concerning the number of rollup batches processed. As shown in Figure 5, the x-axis represents the block gas limits, categorized into 200k Gas, 350k Gas, and 500k Gas, which correspond to caps of approximately 60, 105, and 150 rollup batches, respectively. The y-axis indicates the number of rollup batches processed. The blue bars represent the normal throughput, showing the system's baseline capability under no attack scenarios, while the orange bars indicate the attack throughput. As shown in Figure 5(a), under normal conditions, throughput increases linearly, with the system processing close to 50 batches at 200k gas, close to 100 batches at 350k gas, and peaking at 153 batches at the maximum gas limit of 500k. However, during an RPC address manipulation attack with a probability of attack of 20%, throughput declines sharply, with the system processing only 42 batches at 200k gas (around 14% reduction), 84 batches at 350k gas (around 15% reduction), and 131 batches at 500k gas (around 16% reduction). This degradation underscores the attack's effectiveness in exploiting transaction processing vulnerabilities and reducing system efficiency.

In contrast, when attack probability increases to 40% (Figure 5(b)), the throughput declines significantly more than observed in the 20% case. At 200k gas, the system processes only 34 batches, representing almost 30% reduction from normal throughput. At 350k gas, throughput drops close to 70 batches, a 32% reduction (compared to 16% previously). Finally, at 500k gas, the throughput falls to 110 batches, a staggering 40 batches less processed. This severe degradation demonstrates that the attack's impact scales not only with gas limit but also with adversarial intensity.

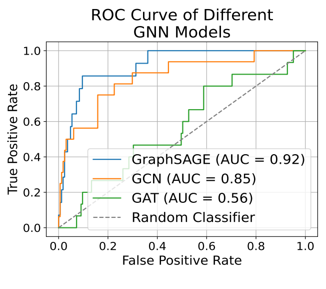
B. Effect on Verifiers' Reward and Slashing

In this section, we analyze how varying attack probabilities (α) impact the average rewards and slashings received by verifiers across different numbers of simultaneous rollup batches in a block, as shown in Figure 6. The graphs compare rewards and slashings under normal conditions and attack scenarios of increasing intensity (α = 0.2, 0.4, 0.6, 0.8). In Figure 6(a), it is observed that under normal conditions, rewards scale linearly with the number of batches, ranging from 11,900 Satoshis to 13,600 Satoshis for 90 batches of the block. However, as α increases, rewards decline significantly. At α = 0.2 average rewards for 90 batches reduce to around 11,000 Satoshis (a 9% decrease), while at α = 0.4, they drop further to 8,500 Satoshis (33% reduction). At higher attack probabilities, the reductions are more pronounced, with average rewards at α = 0.6 and α = 0.8 falling to 6,000 Satoshis (50% reduction) and 2,200 Satoshis (80% reduction), respectively. This trend highlights the vulnerability of verifiers' incentives under intensified adversarial conditions.

Moreover, we analyze the impact of varying attack probabilities on the average slashes incurred by verifiers across different numbers of simultaneous rollup batches in a block. As shown in Figure 6(b), under normal conditions, slashes remain consistently low, peaking at around 1,000 Satoshis for 80 batches. However, as α increases, slashes rise significantly. At α = 0.2, slashes peak at 2,000 Satoshis for 80 batches, doubling the normal values. As attack intensity grows, slashes further escalate, reaching 3,000 Satoshis at α = 0.4, 4,500 Satoshis at α = 0.6, and a severe 6,000 Satoshis at α = 0.8, representing a sixfold increase compared to normal conditions. This trend underscores the compounding penalties verifiers

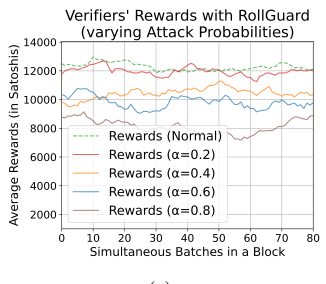


(a)

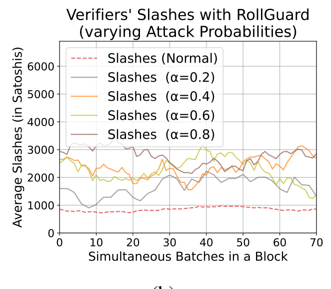


(b)

Fig. 7. Analyzing the training and inference performance of GNN models: (a) training loss curves of GCN, GAT, and GraphSAGE over 40 epochs and (b) ROC curves of GCN, GAT, and GraphSAGE, comparing their anomaly detection performance.



(a)



(b)

Fig. 8. Analyzing the impact of the RollGuard framework in improving verifiers': (a) average reward gained over the batches of a block and (b) average slashing received over the batches of a block (in Satoshis), with varying attack probabilities.

face under heightened attack probabilities, amplifying the risk to their participation and the overall protocol integrity.

C. The Training and Inference of GNN Models

In this section, we evaluate the training and inference performance of RollGuard using three different GNN architectures [18]: **Graph Convolutional Network (GCN), Graph Attention Network (GAT), and GraphSAGE** (Figure 7). As presented in Figure 7(a), the loss values of these models are compared over 50 training epochs to assess their convergence behavior and effectiveness in detecting anomalous RPC address manipulations. Initially, all models exhibit a high loss value of approximately 1.0×10^9, which decreases as training progresses. By epoch 10, GCN and GAT reduce their loss to around 2.5×10^8, while GraphSAGE lags at approximately 4.0×10^8. By epoch 25, GCN stabilizes at a loss of 6.3×10^7, which is lower than GAT's 8.1×10^7 and significantly better than GraphSAGE's 1.2×10^8. These results indicate that GCN achieves faster and more stable convergence.

Later, we analyze the receiver operating characteristic (ROC) curves of RollGuard using GCN, GAT, and GraphSAGE. As shown in Figure 7(b), the ROC curve illustrates the trade-off between the true positive rate (TPR) and the false positive rate (FPR) for each model. The area under the curve (AUC) is used as a key metric to evaluate classification performance. We observe that GraphSAGE achieves the highest AUC of 0.92, indicating superior detection capability for anomalous RPC address manipulations. GCN follows with an AUC of 0.85, demonstrating competitive performance but slightly lower sensitivity to attacks. In contrast, GAT performs significantly worse, with an AUC of only 0.56, barely outperforming a random classifier. These results suggest that GraphSAGE is the most effective model for detecting RPC-based attacks, providing the best balance between detection accuracy and false positive rate.

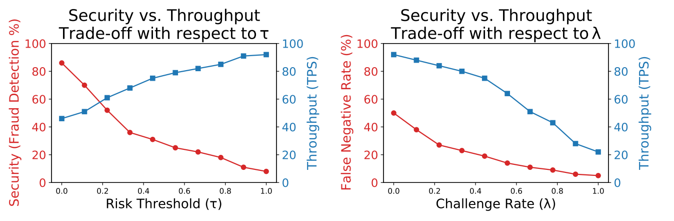
D. Validation of RollGuard's Effectiveness

In this section, we analyze the impact of RollGuard on verifiers' rewards and slashings under different attack probabilities (Figure 8). As shown in Figure 8(a), the average rewards (measured in Satoshis) are plotted against the number of simultaneous batches in a block, considering different attack probabilities ($\alpha = 0.2, 0.4, 0.6, 0.8$). We observe that under normal conditions (i.e., no attack), verifiers receive the highest average rewards, exceeding 12,000 Satoshis having 80 simultaneous batches in the block. However, as the attack probability increases, the rewards decline due to the higher frequency of fraudulent challenges and incorrect verifications. For $\alpha = 0.2$, the rewards remain relatively stable, averaging around 11,800 Satoshis. When the attack probability rises to $\alpha = 0.6$, the rewards drop slightly below 10,000 Satoshis, indicating a negative impact on verifiers' earnings. At the highest attack probability ($\alpha = 0.8$), the rewards deteriorate further, approaching 8,000 Satoshis at higher batch numbers. These results demonstrate that even under high attack probabilities, the rewards with RollGuard remain significantly higher than the scenarios without defense, where the rewards drop drastically to around 2,000 Satoshis. By proactively identifying fraudulent challenges, RollGuard helps sustain payoffs, maintaining system robustness in ensuring fair compensation for verifiers.

In Figure 8(b), the average slashes (measured in Satoshis) are plotted against the number of simultaneous batches in a block, considering similar attack probabilities as before. We observe that under normal conditions, verifiers incur minimal slashes, staying well below 1,000 Satoshis across all batch sizes. As the attack probability increases, slashes rise due to an increase in fraudulent challenges and incorrect verifications. For $\alpha = 0.2$, slashes remain relatively low, averaging around 1,500 Satoshis. However, as the attack probability increases to $\alpha = 0.6$, the slashes reach approximately 2,700 Satoshis, indicating a substantial penalty for verifiers. At the highest attack probability ($\alpha = 0.8$), slashes exceed 3,000 Satoshis, reflecting the significant challenges verifiers face in hostile environments. Notably, even under the most adversarial conditions, RollGuard effectively reduces slashes compared to cases without defense, where slashes escalate beyond 6,000 Satoshis. By proactively detecting anomalous RPC manipulations, RollGuard prevents unnecessary slashes, thereby enhancing the stability and security of the OR systems.

E. Trade-off w.r.t. Risk Threshold and Challenge Rate

In this section, we analyze the impact of the risk threshold (τ) and challenge rate (λ) on the trade-off between security



(a) (b)

Fig. 9. Analysis of the security vs. throughput trade-off for the proposed RollGuard frmaework, governed by: (a) risk threshold (τ) and (b) challenge rate (λ).

and throughput in RollGuard, as shown in Figure 9. First, we experiment with the τ parameter, which is presented in Figure 9(a). The x-axis represents the risk threshold, while the left y-axis represents security, measured as the fraud detection rate percentage, and the right y-axis represents throughput, measured in transactions per second (TPS). As τ increases, security declines while throughput improves. At low values (e.g., $\tau = 0.1$), verifiers are more conservative, issuing challenges frequently, leading to high security (fraud detection near 90%) but at the cost of lower throughput due to increased dispute resolution overhead. Conversely, at high values of τ (e.g., $\tau = 0.9$), RollGuard aggressively avoids challenging against suspected RPC manipulators, allowing transactions to proceed without wrong challenge phases incurred delays, thereby maximizing throughput. However, this also increases the likelihood of fraudulent batches bypassing detection (avoiding actual tampered batches), reducing security. The trade-off curve suggests that an optimal risk threshold ($\tau^*\approx0.18$) exists, balancing security and throughput. Setting τ too low results in unnecessary disputes, limiting system efficiency, whereas setting it too high risks allowing tampered data onto the L1 blockchain, compromising integrity.

Further, we analyze the impact of the challenge rate (λ) on the trade-off between security and throughput. As shown in Figure 9(b), the x-axis represents the challenge rate, while the left y-axis represents security, measured as the false negative rate, and the right y-axis represents throughput, measured in TPS. As λ increases, security improves while throughput declines. At low values (e.g., $\lambda = 0.1$), the system is more lenient, issuing fewer challenges. This results in higher throughput, as transactions proceed with minimal interference, but comes at the cost of reduced security, with an elevated False Negative Rate. Conversely, at high values of λ (e.g., $\lambda = 0.9$), the system adopts stricter security measures, issuing challenges frequently. The trade-off curve implies that an optimal challenge rate ($\lambda^*\approx0.42$) exists, balancing security and throughput. Setting λ too low prioritizes efficiency but risks letting security threats bypass detection. Conversely, setting it too high enhances security at the expense of system efficiency in terms of throuput.

VII. Conclusion

In this work, we identified a critical security vulnerability in the OR systems stemming from RPC address manipulation. This exploit allows attackers to deceive verifiers, triggering false fraud challenges and undermining the integrity of the blockchain protocol. To address this issue, we proposed RollGuard, a novel graph ML-driven security solution leveraging GNNs to dynamically analyze blockchain interactions and detect anomalous RPC address changes in real-time. Our experimental results demonstrate that RollGuard significantly reduces false-positive challenge rates, offering early detection of potential attacks and safeguarding verifiers from undue penalties. By modeling blockchain interactions as a graph and analyzing both real-time and historical transaction patterns, RollGuard provides a proactive and efficient defense mechanism against RPC-based attacks.

References

[1] D. Yang, C. Long, H. Xu, and S. Peng, "A review on scalability of blockchain," in *Proceedings of the 2020 the 2nd International Conference on Blockchain Technology*, 2020, pp. 1–6.
[2] L. Donno, "Optimistic and validity rollups: Analysis and comparison between optimism and starknet," *arXiv:2210.16610*, 2022.
[3] L. T. Thibault, T. Sarry, and A. S. Hafid, "Blockchain scaling using rollups: A comprehensive survey," *IEEE Access*, 2022.
[4] G. Picco and A. Fortugno, "Ephemeral rollups are all you need," *arXiv preprint arXiv:2311.02650*, 2023.
[5] Z. Cheng, X. Hou, R. Li, Y. Zhou, X. Luo, J. Li, and K. Ren, "Towards a first step to understand the cryptocurrency stealing attack on ethereum," in *22nd international symposium on research in attacks, intrusions and defenses (RAID 2019)*, 2019, pp. 47–60.
[6] C. R. Team, "Endpoint secrets exposed: A tale of 8000 dapps," https://chainstack.com/assets/Endpoint
[7] K. Li, J. Chen, X. Liu, Y. R. Tang, X. Wang, and X. Luo, "As strong as its weakest link: How to break blockchain dapps at rpc service." in *NDSS*, 2021.
[8] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.
[9] T. Schaffner, "Scaling public blockchains," *A comprehensive analysis of optimistic and zero-knowledge rollups. University of Basel*, 2021.
[10] H. S. de Ocáriz Borde, "An overview of trees in blockchain technology: merkle trees and merkle patricia tries," *University of Cambridge: Cambridge, UK*, 2022.
[11] Optimism-Documentation, "Mempool," https://community.optimism.io/docs/developers/bedrock/differences-mempool.
[12] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.
[13] S. S. Team, "Unveiling a new scam: Malicious modification of rpc node links to steal assets," *SlowMist Blog*, 2023. [Online]. Available: https://slowmist.medium.com/unveiling-a-new-scam-malicious-modification-of-rpc-node-links-to-steal-assets-2ca324200853
[14] Z. Chang, Y. Cai, X. F. Liu, Z. Xie, Y. Liu, and Q. Zhan, "Anomalous node detection in blockchain networks based on graph neural networks," *Sensors*, vol. 25, no. 1, p. 1, 2024.
[15] A. Sharma, P. K. Singh, E. Podoplelova, V. Gavrilenko, A. Tselykh, and A. Bozhenyuk, "Graph neural network-based anomaly detection in blockchain network," in *International Conference on Computing, Communications, and Cyber-Security*. Springer, 2022, pp. 909–925.
[16] Optimism, "Node snapshots documentation," https://docs.optimism.io/builders/node-operators/management/snapshots, n.d., accessed: 2025-01-04.
[17] Internet Archive, "Optimism snapshot archive," https://archive.org/download/optimism-snapshot, n.d.
[18] D. Firdaus and I. Sumardi, "Comparative analysis of graph neural network with sage conv, gat conv, and gcn conv techniques for fake news detection," *Industrial Sciencetech Journal*, 2024.