Deanonymizing Ethereum Users behind Third-Party RPC Services

Shan Wang[§], Ming Yang[§]*, Wenxuan Dai[§], Yu Liu[§], Yue Zhang[†], Xinwen Fu[‡]
[§] School of Computer Science and Engineering, Southeast University.
Email: shanwangsec@gmail.com, {yangming2002, dwx, liuyu_}@seu.edu.cn.
[†]Drexel University. Email: yz899@drexel.edu.
[‡]University of Massachusetts Lowell. Email: Xinwen_Fu@uml.edu.

IEEE INFOCOM 2024 - IEEE Conference on Computer Communications | 979-8-3503-8350-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/INFOCOM52122.2024.10621236

***Abstract*—Third-party RPC services have become the mainstream way for users to access Ethereum. In this paper, we present a novel deanonymization attack that can link an Ethereum address to a real-world identity such as IP address of a user who accesses Ethereum via a third-party RPC service. We find that RPC API calls result in distinguishable sizes of encrypted TCP packets. An attacker can then find when a user sends a transaction to an RPC provider and immediately send a beacon transaction after the user transaction. By exploiting the differences in the distributions of inter-arrival time intervals of normal transactions and two simultaneously initiated transactions, the attacker can identify the victim transaction in the Ethereum network. This enables the attacker to correlate the Ethereum address of the victim transaction's initiator with the source IP address of TCP packets from a victim user. We model the attack through empirical measurements and conduct extensive real-world experiments to validate the effectiveness of our attack. With three optimization strategies, the correlation accuracy can reach to 98.70% and 96.60% respectively in Ethereum testnet and mainnet. We are the first to study the deanonymization of Ethereum users behind third-party RPC services.**

***Index Terms*—Ethereum, Deanonymization, RPC Service**

I. Introduction

The mainstream approach for users to access Ethereum networks is now through third-party RPC (Remote Procedure Call) services. For example, *Infura* is a representative third-party RPC provider with over 400,000 developers [1]. In 2021, about 4.8 trillion Ethereum transactions were sent from *Infura* [2]. A cryptocurrency wallet application named *MetaMask* is developed upon *Infura*, and has more than 21 million active users in 2023 [3]. With RPC services, users can run a lightweight wallet application or a script to initiate transactions or query blockchain states by sending RPC requests.

Existing deanonymization attacks against blockchains cannot identify the real-world identity, such as the IP address, of a user accessing Ethereum via a third-party RPC service. Blockchain addresses may be clustered to know that a cluster of blockchain addresses belongs to an individual user [4]–[8]. However, the real-world identity of the user cannot be identified. The IP address of the origin node that sends a transaction to the blockchain network may also be identified [9]–[11]. However, such a method cannot be applied to users who use the same third-party RPC service since the origin node of a transaction is the server of the RPC provider in this context.

* Corresponding author: Prof. Ming Yang of Southeast University, China.

In this paper, we present a novel deanonymization attack that can determine the IP address of an Ethereum user using a third-party RPC service. We assume that an attacker can monitor and send network traffic going through a network gateway such as a router. This assumption is popular in studying various network attacks [12]–[14]. The attacker also deploys a node in Ethereum as a probe, which collects the arrival times of transactions. A victim user connects to the Internet through the gateway, and accesses Ethereum via a third-party RPC provider such as *Infura*. By carefully analysing the varying semantics of all RPC APIs, we find that the size ranges of TCP segments from a user can be exploited to pinpoint TCP packets used to initiate a transaction. Once such a victim transaction is identified, the attacker initiates its own transaction as a *beacon* through the same RPC service. The attacker's probe node in Ethereum keeps recording arrival times of transactions including the beacon transaction. We find that the arrival time of the beacon transaction can be used to estimate the arrival time of the victim transaction through statistics of transaction inter-arrival time intervals. The insight is that the inter-arrival time interval of the victim transaction and beacon transaction is statistically different from the distribution of a pair of normal transactions in the Ethereum network.

Our major contributions can be summarized as follows. (i) We are the first to study the deanonymization of Ethereum users who use third-party RPC services. (ii) We model the attack theoretically to capture factors that affect the attack. (iii) Real-world experiments are conducted in both Ethereum testnet *Goerli* and mainnet to validate our theoretical results and the attack effectiveness in practice. By observing one transaction from a victim IP, the attacker can achieve an accuracy of 95.40% in testnet and 90.00% in mainnet. With multiple transactions from a victim IP, it can further employ *multiple confirmations*. The accuracy can reach 98.70% in testnet and 96.60% in mainnet via averaging two confirmations. (iv) Our attack is of low cost as it only requires transaction fees for initiating beacon transactions. There is no charge in testnet. If a user is active in both mainnet and testnest, she/he can be identified in testnest with no fees.

Ethical Considerations. We collect only transaction arrival times on the Ethereum testnet and mainnet and examine their statistical features without interfering with the normal operation of these networks or exposing the real-world identities

of their users. The attack experiments were conducted against transactions initiated by us, revealing our own IP addresses.

II. Background

This section introduces Ethereum address and transaction, RPC services, and transaction propagation mechanisms.

A. Ethereum Address and Transaction

Each Ethereum user and smart contract is uniquely identified by a 42-character address. The address performs as the identity of a user or a smart contract in the Ethereum network, and is used as the source address or destination address to create or receive a transaction. Such an address is derived from the hash of a public key, which is not linked to any real-world identity. With the pseudonymity mechanism, Ethereum enables users to conceal their real-world identities, providing a degree of anonymity and privacy protection.

An Ethereum transaction contains several fields. Each transaction is initiated and signed by a user with an Ethereum address, and the *From* field indicates the Ethereum address of its initiator user. Field *To* is the address of the transaction recipient which can be a user or a smart contract. In a smart contract deployment transaction, field *To* is null. *Value* field contains the amount of native tokens to be transferred in this transaction. *Data* field is optional and can be arbitrary data. If a transaction is used to deploy or invoke a smart contract, *Data* will carry the smart contract code or the name and parameters of a function to be invoked. If a transaction only transfers tokens, *Data* field can be null. Field *GasLimit* specifies the gas amount that is allowed to be consumed by the transaction. The gas price for a transaction is defined by the transaction initiator through the field *maxPriorityFeePerGas*, which specifies the fee that the initiator is willing to pay for executing the transaction. A transaction only reveals the Ethereum address of its initiator, not its real-world identity.

B. Ethereum RPC Service

Ethereum nodes support RPC protocol, through which users can conveniently interact with the Ethereum network by sending RPC requests and receiving RPC response. As shown in Fig. 1, a user can interact with a third-party RPC provider by running a lightweight wallet application like *MetaMask* or a *Web3* script on their low-end personal devices. An RPC provider maintains its own full nodes in Ethereum, and opens RPC APIs to its users. To send a transaction, a user can propose an RPC request to invoke an API named *sendRawTransaction*. This request contains a raw transaction which is signed by the user. Then the RPC provider can convert the request into a well-formatted transaction, and propagate the transaction to Ethereum. An RPC provider also synchronizes blocks and transactions in Ethereum, and provides APIs such as *getBalance* for users to query the blockchain states.

C. Transaction Propagation

In the Ethereum P2P network, a transaction propagates from an origin node to other nodes. The origin node either creates the transaction or receives the transaction from an off-chain source. In the third-party RPC service scenario, a node of

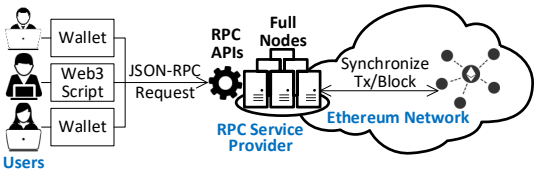

Fig. 1. Third-Party RPC Service in Ethereum

the RPC provider acts as the origin node. Initially, the origin node propagates the transaction to its neighbor nodes. Upon receiving a transaction, a node validates the transaction, stores the transaction in its transaction pool if the transaction settings such as gas price meet its self-defined standards, and relays pooled transactions to its neighbors. Please note that, a node will not relay a transaction to its neighbors who already know this transaction, such as a neighbor who previously sends this transaction to the node. Finally, a valid transaction should appear in the transaction pool of each node.

A transaction with low gas price may experience abnormal propagation. Each node has a size limit on its transaction pool, and prioritizes transactions with high gas price. A node may drop a transaction if its gas price does not meet the node's standards. Additionally, a miner node may not prioritize picking up a pooled transaction with low gas price to wrap a new block. Transactions that remain pending in a pool for a period may eventually be dropped. Determining an appropriate gas price is challenging because it fluctuates with the level of blockchain network congestion and the number of active users. A common way for a user to get an appropriate gas price is to query a suggested gas price from an RPC service.

III. Attack Overview

In this section, we introduce the system model and threat model, analyze attack goals and challenges, and present the basic idea and insights of the proposed attack.

A. System Model and Threat Model

As shown in Fig. 2, a group of users use their personal devices to connect to the *Internet* through a gateway, and access Ethereum via a third-party RPC service. When a user sends an RPC request, the corresponding network data flows through the gateway. The RPC protocol operates over the application-layer protocol *HTTPS*, which runs on top of the transport-layer TCP protocol and adopts TLS protocol to encrypt the transmitted data between the user and the RPC provider.

We assume that an attacker can monitor the network traffic going through the gateway, as well as can send network data through this gateway. This is a normal assumption in network attacks [12]–[14]. The attacker may be the owner of the gateway, an internet service provider (ISP), or a regulatory authority. The attacker also deploys a node in the public Ethereum as a probe, which receives propagated transactions and collects their arrival time. A user who interacts with an RPC provider through the gateway is a victim. We refer to a transaction initiated by the victim user as a victim transaction, which is transmitted in victim TCP packets that are encrypted by TLS protocol. We assume an RPC provider is trustworthy.

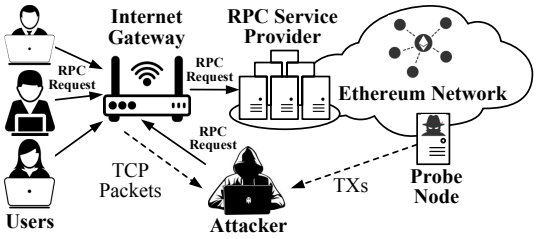

Fig. 2. System Model and Threat Model

B. Attack Goals and Challenges

With the network data collected from the gateway and the transaction data collected from the probe node, an attacker aims to correlate a victim user's Ethereum address with the IP address of the user's personal device, which can be considered as a real-world identity of the user [15]. The attacker mainly faces two challenges as follows.

Encrypted TCP Packets. The TCP packets that may carry a raw transaction between a victim user and an RPC provider are encrypted by the TLS protocol. Although an attacker can obtain the source IP address of TCP packets, which may come from a victim user, it is challenging for the attacker to identify TCP packets that pertain to sending a transaction to an RPC provider, or to obtain the raw transaction and its initiator's Ethereum address from the TCP packets.

Enormous Normal Transactions. A significant number of users are active in Ethereum network and initiate transactions. It is challenging to pinpoint the transaction initiated by a victim user from the vast number of daily transactions flowing in the entire Ethereum network. Existing work [9]–[11] can infer the IP address of the origin node of a transaction. However, in the third-party RPC service context, the origin node belongs to an RPC provider, and existing methods cannot correlate the victim user's IP address with a transaction.

C. Basic Idea and Insights

We propose a novel deanonymization attack method that overcomes the aforementioned two challenges as follows.

Victim TCP Packets Identification (§IV). An attacker exploits TCP packets to identify when a victim user with a certain IP address is sending a victim transaction to an RPC provider. The attack first exploits the TLS handshake mechanism to identify TCP packets that are directed towards a certain RPC provider, where the domain name of the RPC provider serves as a strong identifier. A provider may offer dozens of RPC APIs. Then the TCP segment sizes are used to pinpoint TCP packets for calling an RPC API named *sendRawTransaction*, which is used to initiate a transaction. The *insight* behind this approach is that the semantics of *sendRawTransaction* is distinguishable from other RPC APIs, whose request contains a relatively large raw transaction while the response only contains a transaction hash with a fixed and small size.

Victim Transaction Identification (§V). Once the attacker observes the victim TCP packets, it immediately initiates a transaction as a *beacon* by sending a request to the same RPC provider as the victim user. The attacker's probe node keeps recording the arrival time of each transaction including the beacon transaction. Using the arrival time of the beacon transaction, the attacker infers the arrival time of the victim transaction since the attacker knows when the victim transaction is sent out through the gateway. The attacker then finds out the victim transactions from the daily transactions on Ethereum using statistical features of inter-arrival time intervals. The *insight* behind this method is that the inter-arrival time interval distribution of two simultaneously initiated transactions is different from the distribution of daily transactions in the entire Ethereum network.

Using the information obtained through the above two steps, the attacker can correlate the source IP address of the identified TCP packets, i.e., IP address of a victim user, with the Ethereum address of the victim transaction's initiator.

IV. Victim TCP Packets Identification

In this section, we illustrate how to accurately identify TCP packets for sending a transaction to a third-party RPC provider.

A. Identifying TCP Packets Related to RPC Services

We exploit the TLS handshake mechanism to identify TCP packets related to a third-party RPC service. The RPC protocol runs over TCP protocol and adopts TLS protocol to secure the transmitted data. Once a TCP connection, identified by 4-tuple (*source IP*, *source port*, *destination IP*, *destination port*), is established, the TLS handshake is initiated to negotiate cipher suites. At the beginning of the TLS handshake, the user/client sends a *Client_Hello* message to the destination server, i.e., RPC provider, which includes a field *Sever_Name* that indicates the domain name of the RPC provider. We collect the domain names of representative RPC providers such as *Infura*, *Alchemy* and *OnFinality*, and use them as strong identifiers to identify the TCP packets related to a third-party RPC service and the RPC provider that a victim user registers with.

B. Identifying TCP Packets for Sending Transactions

RPC providers mainly support 44 types of RPC APIs in total [16], and we need to pinpoint TCP packets for calling *sendRawTransaction* API which is used to initiate a transaction, while other APIs are mainly used to query the blockchain state and do not generate a transaction.

We analyse features of TCP segment sizes that correspond to different RPC APIs. A TCP segment in TCP packets contains the ciphertext of an RPC request/response and some TLS cipher characters of a fixed size. Considering the application-layer protocols HTTP/1 and HTTP/2, an RPC request/response contains a JSON object and a HTTP header. A JSON object in an RPC request or response contains the parameters or returned results of an RPC API, and its size varies upon the API semantics. Please note that, application-layer protocols are distinguishable per their communication patterns [17], and we do not consider the uncommon *WebSocket* protocol in RPC.

JSON Object. For each of the 44 RPC APIs, we carefully analyze its semantics and calculate the size range of its request and response JSON object respectively. Take the *sendRawTransaction* API as an example. Its request JSON contains four key-value pairs. The values of keys *jsonrpc* and *method* are fixed. The value of key *id* is a random number,

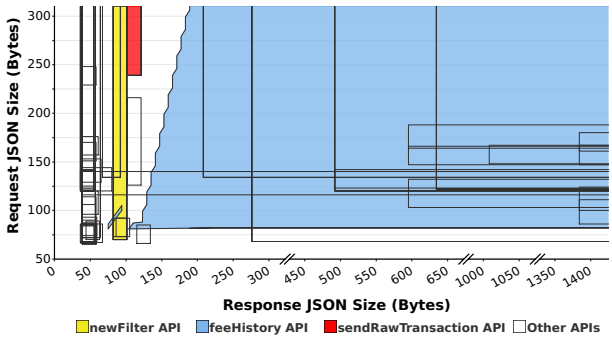

Fig. 3. Area Range of Request/Response JSON Object Size for 44 RPC APIs.

whose size ranges from 1 to 20 bytes. The value of key *params* is a raw transaction. The smallest raw transaction should be a contract deployment transaction whose both *To* field and *Data* field are set to null. In theory, the block gas limit is the most gas that a single transaction can consume, which indirectly limits the transaction size. We estimate the largest size of a raw transaction according to the block gas limit of 30.00 million gas on July 1st, 2023. The response JSON of *sendRawTransaction* also includes key-value pairs with keys *jsonrpc* and *id*, as well as a key-value pair with the key *result*. The value of *result* is the hash of the initiated transaction, whose size is fixed at 66 bytes. Upon the above analysis, we derive that the size of *sendRawTransaction* request JSON ranges from 239 to about 14.99 million bytes, while the size of its response JSON ranges from 102 to 121 bytes.

Fig. 3 shows the area range of sizes of request and response JSON objects for each of the 44 RPC APIs, and the area range of *sendRawTransaction* in red does not overlap with any other APIs. The blue area represents API *feeHistory* [16], which may have the same response JSON size as *sendRawTransaction*. However, with the same response JSON size, the request JSON size of *feeHistory* is smaller than that of *sendRawTransaction*, since *sendRawTransaction* accepts a relatively large raw transaction as input and returns a transaction hash with fixed 66 bytes. The yellow area represents *newFilter* API, which returns a filter ID with a fixed size. Its response JSON size ranges from 82 to 101 bytes, making its area to be just at the left of the area of *sendRawTransaction*, with a gap of only 1 byte.

HTTP Header. The fields in a HTTP header may vary per different settings, but will not lead to area range overlap between *newFilter* and *sendRawTransaction* in most cases. The size of a HTTP header may vary due to a varied *length* field indicating the length of the JSON object, a *date* field whose size may vary due to the HPACK header compression, and some other fields per application settings such as fields for method name, trace ID and so on. We carefully analyse the HTTP settings of three representative RPC providers including *Infura*, *Alchemy*, and *OnFinality*, and calculate the size ranges of their HTTP header. As summarized in Table I, *sendRawTransaction* and *newFilter* can keep distinguishable upon the RPC request and RPC response TCP segment sizes in most cases. Only in the *Infura* with HTTP/2 case, the response TCP segment size of *newFilter* overlaps with that of *sendRawTransaction* by 2 bytes.

Identification Rules based on TCP Segment Size. We use the request/response TCP segment size range of *sendRawTransaction* as rules to identify its related TCP packets according to Table I. Specially, in the case of *Infura* with HTTP/2, the response TCP segment size of 250 or 251 bytes may correspond to both *sendRawTransaction* and *newFilter* due to their overlap, potentially leading to false positives. To avoid false positives, we define the identification rule in this case as: **(i) the response TCP segment size ranges from 252 to 271 bytes**, and **(ii) the request TCP segment size is equal to or larger than 391 bytes.** This will result in a false negative rate of about 9.10%, while maintaining a true positive rate of 100%. In other cases, the false negative rate is 0, and the true positive rate is 100% as summarized in Table I.

V. Victim Transaction Identification

In this section, we demonstrate how to utilize inter-arrival time intervals of transactions to find out the victim transaction, which is generated through the identified victim TCP packets.

A. Inter-Arrival Time Interval of Transactions

An attacker deploys a node in Ethereum as a probe, and collects arrival time of each received transaction. As analysed in Sec. II-C, a node will not relay a transaction to its neighbors who have known this transaction. Therefore, we customize the probe node to only receive transactions, but not relay the received transactions to its neighbors. In this way, each neighbor should relay a transaction to the probe node, if the transaction gas price satisfies standards of these nodes. Assume the probe node has n neighbors. For each transaction Tx_i, the probe node records the arrival time t_{ij} when it receives Tx_i from j-th neighbor, and obtains an arrival time vector $\langle t_{i1}, t_{i2}, ..., t_{in} \rangle$.

In practice, the dimension of arrival time vectors for a pair of transactions may differ, as they may be relayed to the probe by different neighbors for two reasons. First, different nodes may set different standards on the gas price, and a low gas price may discourage most neighbor nodes from relaying a transaction. Second, neighbor nodes may vary as they may unpredictably disconnect from the probe node. To calculate the inter-arrival time interval of two transactions, we only consider a *common vector* for each transaction by taking the arrival time corresponding to *common* neighbors who relay both transactions. For example, assume the arrival time vector of Tx_1 is $\langle t_{11}, t_{12}, t_{13}, t_{15} \rangle$, and that of Tx_2 is $\langle t_{21}, t_{22}, t_{23}, t_{26} \rangle$. Then we create a 3-dimensional common vector for each transaction, i.e., $\langle t_{11}, t_{12}, t_{13} \rangle$ and $\langle t_{21}, t_{22}, t_{23} \rangle$, respectively.

We use the common vector of m dimensions to calculate the interval of two transactions such as Tx_i and Tx_j. We first calculate the interval in each dimension and get $\langle t_{i1} - t_{j1}, t_{i2} - t_{j2}, ..., t_{im} - t_{jm} \rangle$. Then we calculate the mean of these m intervals, which is considered as the interval I_{ij} of transactions Tx_i and Tx_j as follows,

$$I_{ij} = \frac{\sum_{k=1}^{m} |t_{ik} - t_{jk}|}{m}. \quad (1)$$

TABLE I
TCP segment size range per different RPC providers

RPC Provider	Protocol	Type	API	HTTP Header				JSON	TLS Cipher	Overall	False Negative
				length	date	others	Overall				
Infura	HTTP/1	request	sendRawTransaction	21-26	0	147	168-173	$\geq$ 239	29	$\geq$ **436**	0
			newFilter	$\geq$ 20	0	147	$\geq$ 167	$\geq$ 70	29	$\geq$ 266	
		response	sendRawTransaction	21	37	112	170	102-121	29	**301-320**	
			newFilter	20-21	37	112	169-170	82-101	29	280-300	
	HTTP/2	request	sendRawTransaction	6-11	0	88-92	94-103	$\geq$ 239	58	$\geq$ **391**	9.10%
			newFilter	$\geq$ 5	0	88-92	$\geq$ 93	$\geq$ 70	58	$\geq$ 221	
		response	sendRawTransaction	5	23-25	62	90-92	102-121	58	**250-271**	
			newFilter	4-5	23-25	62	89-92	82-101	58	229-251	
Alchemy	HTTP/1	request	sendRawTransaction	21-26	0	156	177-182	$\geq$ 239	22	$\geq$ **438**	0
			newFilter	$\geq$ 20	0	156	$\geq$ 176	$\geq$ 70	22	$\geq$ 268	
		response	sendRawTransaction	21	37	687	745	102-121	22	**869-888**	
			newFilter	20-21	37	687	744-745	70-89	22	836-856	
	HTTP/2	request	sendRawTransaction	6-11	0	93-105	99-116	$\geq$ 239	44	$\geq$ **382**	0
			newFilter	$\geq$ 5	0	93-105	$\geq$ 98	$\geq$ 70	44	$\geq$ 212	
		response	sendRawTransaction	5	23-25	117-124	145-154	102-121	44	**291-319**	
			newFilter	4-5	23-25	117-124	144-154	70-89	44	258-287	
OnFinality	HTTP/1	request	sendRawTransaction	21-26	0	123	144-149	$\geq$ 239	22	$\geq$ **405**	0
			newFilter	$\geq$ 20	0	123	$\geq$143	$\geq$ 70	22	$\geq$ 235	
		response	sendRawTransaction	21	37	215	273	102-121	22	**397-416**	
			newFilter	20-21	37	215	272-273	70-89	22	364-384	
	HTTP/2	request	sendRawTransaction	6-11	0	72	78-83	$\geq$ 239	44	$\geq$ **361**	0
			newFilter	$\geq$ 5	0	72	$\geq$ 77	$\geq$ 70	44	$\geq$ 191	
		response	sendRawTransaction	5	23-25	141-145	169-175	102-121	44	**315-340**	
			newFilter	4-5	23-25	141-145	168-175	70-89	44	282-308	

B. Basic Attack Method

When an attacker identifies TCP packets for sending a victim transaction Tx_v as introduced in Sec. IV, the attacker immediately initiates a *beacon* transaction Tx_b to the same RPC provider. To avoid considering abnormal propagation caused by low gas price, the attacker initiates a beacon transaction with suggested gas price, and only considers transactions whose common vector has a dimension of no less than D when comparing it to the beacon transaction.

Assume the time of observing victim transaction is t_v, and the time of initiating beacon transaction is t_b. The delta is $\triangle = t_b - t_v$. The probe node obtains the arrival time vector of beacon transaction as $\langle t_{b1}, t_{b2}, ..., t_{bm} \rangle$. Intuitively, Ethereum P2P network maintains a stable topology within a small time window, and transactions from the same RPC provider are likely to propagate through the same paths in the P2P network. Therefore, if the beacon transaction is sent at the same time as observing Tx_v, i.e., at t_v, its arrival time vector may be $\langle t_{b1} - \triangle, t_{b2} - \triangle, ..., t_{bm} - \triangle \rangle$, which represents a beacon transaction $Tx_b^{'}$ simultaneously initiated with the victim transaction Tx_v.

Intuitively, two transactions initiated simultaneously by the same RPC provider should have a small inter-arrival time interval. Therefore, we consider a transaction as a potential victim transaction if its interval to transaction $Tx_b^{'}$ is within a time window T. There may be several potential victim transactions that form a transaction set S_{pot}. Assume the number of transactions in set S_{pot} is n. There are only three possible cases per the relationship between Tx_v and S_{pot} , as follows.

Case 1: $Tx_v \notin S_{pot}$. The victim transaction is not in the potential transaction set, which indicates a **false identification**.

Case 2: $Tx_v \in S_{pot}$ and $n = 1$. The victim transaction is uniquely identified, indicating an **accurate identification**.

Case 3: $Tx_v \in S_{pot}$ and $n > 1$. The victim transaction is included in the potential transaction set S_{pot}, but other transactions also appear in S_{pot}. In this case, the victim transaction cannot be uniquely and accurately identified, we consider it a case of **non-unique identification**.

The probability of each of the three cases may be influenced by two main factors. (i) The number of transactions from other users and nodes that arrive at the probe node with an interval to $Tx_b^{'}$ within T. (ii) Whether two simultaneously initiated transactions by an RPC provider will arrive at the probe node with an interval within T.

C. Modeling

We formalize the probability of the three cases by modeling the transaction inter-arrival time interval distribution.

We first model the inter-arrival time interval of daily transactions in the entire Ethereum network. We use Poisson

distribution to model the probability of a given number k of transactions arriving at the probe node in a fixed interval of time t as follows,

$$P(t,k) = \frac{(\frac{\lambda t}{\delta})^k \ e^{-(\frac{\lambda t}{\delta})}}{k!}, \tag{2}$$

where λ is the average number of transactions that arrive at the probe node within a time unit δ. Following the Poisson distribution, the distribution of transaction inter-arrival time interval t should be a exponential distribution as follows,

$$P_{inter}(t) = e^{-(\frac{\lambda t}{\delta})}. \tag{3}$$

We further use a normal distribution to model the inter-arrival time interval, i.e., I_{sim}, of two simultaneously initiated transactions through the same RPC provider, i.e., $I_{sim} \sim N(\mu, \rho^2)$. The probability of two simultaneous transactions having an interval of t is as follows,

$$F(t) = \frac{1}{\rho\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{t-\mu}{\rho})^2}, \tag{4}$$

where μ is the expectation of the interval I_{sim} and ρ is the standard deviation. The cumulative probability of $I_{sim} \leq t$ is,

$$\Phi(t) = \int_{-\infty}^{t} F(t)dt. \tag{5}$$

Ideally, two simultaneously initiated transactions should arrive at the probe node at the same time and the interval is 0. Therefore, we set the expectation of interval I_{sim} as $\mu = 0$.

Considering the interval should be an absolute value of the delta between two arrival time common vectors as shown in Equation (1), we model the absolute interval $I_{sim}^{'}$ on top of the normal distribution $N(\mu, \rho^2)$. The probability of two simultaneous transactions having an absolute interval of t is as follows,

$$F^{'}(t) = 2 * F(t), \ t \geq 0. \tag{6}$$

The cumulative probability of $I_{sim}^{'} \leq t$ is as follows,

$$\Phi^{'}(t) = \Phi(t) - \Phi(-t), \ t \geq 0. \tag{7}$$

With the Poisson distribution model and the normal distribution model, we formalize the probability of Case 1—false identification, Case 2—accurate identification and Case 3—non-unique identification, as follows.

Case 1: $Tx_v \notin S_{pot}$. This case means that the interval between Tx_v and $Tx_b^{'}$ is larger than the time window T. According to Equation (7), the false identification rate is,

$$P(Tx_v \notin S_{pot}) = 1 - \Phi^{'}(T). \tag{8}$$

Case 2: $Tx_v \in S_{pot}$ and $n = 1$. This case means that Tx_v arrives at the probe node with an interval to $Tx_b^{'}$ within T, and there is no other transaction arrives at the probe node. According to Equation (2) and (7), the accuracy rate is,

$$P(Tx_v \in S_{pot}, n = 1) = \Phi^{'}(T) * P(T, 0). \tag{9}$$

Case 3: $Tx_v \in S_{pot}$ and $n > 1$. This case means that Tx_v arrives at the probe node with an interval to $Tx_b^{'}$ within T, but other transactions also arrive at the probe node. According to Equation (2) and (7), the non-unique identification rate is,

$$P(Tx_v \in S_{pot}, n > 1) = \Phi^{'}(T) * P(T, k \geq 1), \tag{10}$$

where $P(T, k \geq 1) = 1 - P(T, 0)$.

D. Optimization Strategies

In Case 3—non-unique identification, the victim transaction Tx_v is already in the potential transaction set S_{pot}, which also includes other transactions. Assume the potential transaction set S_{pot} contains m transactions $\{Tx_1^{pot}, Tx_2^{pot}, ..., Tx_m^{pot}\}$, and Tx_1^{pot} is the victim transaction. We design three strategies to further eliminate other transactions, so as to uniquely identify the victim transaction and improve the accuracy.

Smallest Interval. Assume an attacker observes only one transaction from a certain IP address. The attacker calculates the interval of Tx_i^{pot}, $i = 1, 2, ..., m$ to transaction $Tx_b^{'}$, and chooses the transaction with the smallest interval as the final identification result.

Multiple Confirmations. Assume an attacker observes multiple transactions from a certain IP address. Each time the attacker gets a potential transaction set S_{pot}, it also gets a set of Ethereum address of initiators of these transactions. The attacker counts the frequency of each address and selects the most frequent ones. When only one Ethereum address is the most frequent address, it is considered as the final result.

Raw Transaction Size. Raw transaction size can be exploited to eliminate false transactions from the set S_{pot}. In the RPC request TCP segment of *sendRawTransaction*, the JSON object only contains a raw transaction. According to Table I, an attacker can estimate the size range of the raw transaction based on the TCP segment size. When a probe node receives a transaction, it can obtain the raw transaction size of each transaction. If the raw transaction size of a potential transaction is out of the estimated size range, the attacker drops it from S_{pot}. This strategy may not find out a unique result, but can be combined with the other two strategies to improve accuracy.

VI. Empirical Measurements

In this section, we empirically measure and validate the formalization models of our attack in both Ethereum testnet and mainnet. Then we derive the theoretical attack results.

A. Data Collection and Preprocessing

We deploy a probe node that maintains 25 connections to other nodes in both Ethereum mainnet and testnet *Goerli*, and collects a total of $9,434,025$ transactions in mainnet and $5,272,540$ transactions in testnet from July 1 to July 7, 2023. We observe that a neighbor may relay a transaction much later, such as several seconds later, than other neighbors, which is abnormal since a transaction normally is broadcast to the entire network within 300 ms [18]. To eliminate abnormal propagation time, we discard any time in a transaction arrival time vector that deviates from other times by more than 300 ms.

We count the dimensions of common vectors for each pair of adjacent transactions respectively in testnet and mainnet. As shown in Fig. 5, in the mainnet, the common vectors of most pairs of transactions have either about 1 dimension or about 13 dimensions. Upon a detailed analysis of transactions with low-dimensional common vectors, we find that the underlying cause was a low gas price. As shown in Fig. 4, in the testnet, only a few pairs of transactions have low-dimensional common vectors. This is reasonable since the gas

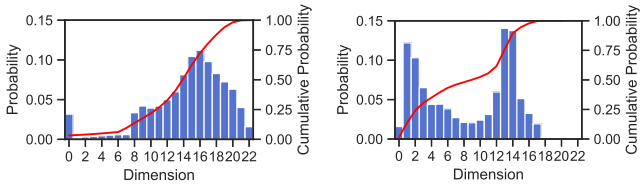

Fig. 4. Dimension Number of Common Vectors in Testnet

Fig. 5. Dimension Number of Common Vectors in Mainnet

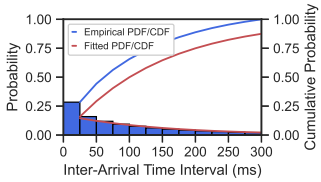

Fig. 6. Inter-arrival Time Interval of Normal Transactions in Testnet

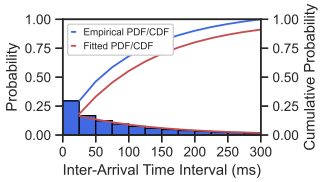

Fig. 7. Inter-arrival Time Interval of Normal Transactions in Mainnet

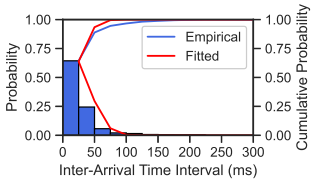

Fig. 8. Inter-arrival Time Interval of Two Simultaneously Initiated Transactions in Testnet

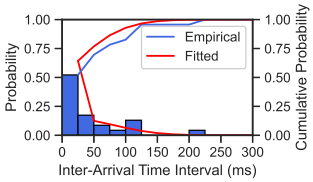

Fig. 9. Inter-arrival Time Interval of Two Simultaneously Initiated Transactions in Mainnet

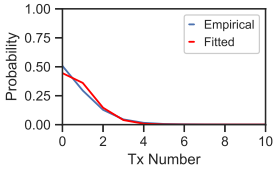

Fig. 10. Poisson Distribution When $t = 100$ ms in Mainnet

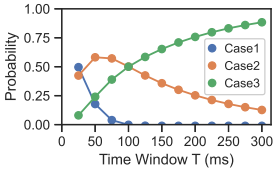

Fig. 11. Theoretical Results of Victim Transaction Identification in Testnet

price required by the testnet is lower than that by the mainnet, and most transactions can be normally relayed. To avoid considering transactions with abnormal propagation caused by low gas prices, and to model the testnet and mainnet in the same way, we only consider transaction pairs whose common vector dimensions are no less than 8, i.e., $D \geq 8$.

B. Model Measurement and Validation

Using the preprocessed data, we first empirically measure and validate the Poisson distribution model for arrival time of daily transactions. Fig. 6 and Fig. 7 respectively show the transaction inter-arrival time interval distribution in testnet and mainnet. When we consider common vectors with 8 or more dimensions, $4,409,370$ transactions remain in testnet, and $4,880,577$ transactions remain in mainnet, since about 48.11% mainnet transactions incur low-dimensional common vectors. When we set the time unit as $\delta = 25$ ms, the average number of arriving transactions is $\lambda = 0.17$ in testnet and $\lambda = 0.20$ in mainnet. We compare the fitted Poisson distribution model in Equation (3) with the empirical measurements. As shown in Fig. 6 and Fig. 7, it can be observed that the fitted inter-arrival time interval distribution is similar to the empirical results with minor bias. Considering the Ethereum network is usually more active at GMT day time than at GMT night time, but Poisson distribution models the arrival time with a fixed average rate, we argue that the bias is reasonable. We also do p-value testing on the fitted Poisson distribution $P(t,k)$ with different t, such as Fig. 10 shows the case when $t = 100$ ms, and it can pass the p-value testing.

We also empirically measure the inter-arrival time interval of two simultaneously initiated transactions through the same RPC provider. We initiate 1000 pairs of transactions through *Infura* in the testnet. As shown in Fig. 8, most pairs of transactions have an interval within 25 ms. We calculate that the mean of I_{sim} is about -1.52 which is close to the expectation $\mu = 0$, and its standard deviation is about 38.39. We set $\rho = 38.39$ in testnet. The red lines in Fig. 8 represent Equations (6) and (7). It shows that the fitted model in red lines is close to the empirical measurements in blue. We also initiate 30 pairs of transactions in the mainnet, and get the empirical mean is about -5.26 and the standard deviation is $\rho = 68.23$. As shown in Fig. 9, the fitted model is close to the empirical measurements with minor bias, which is reasonable since the data samples in mainnet are limited.

C. Theoretical Results

Overall, the inter-arrival time intervals of daily transactions across the entire Ethereum network follow a Poisson distribution $P(t,k)$ with $\lambda = 0.17$ in testnet and $\lambda = 0.20$ in mainnet when $\delta = 25$ ms. The inter-arrival time intervals of two simultaneously initiated transactions follow a normal distribution with $\mu = 0$ and $\rho = 38.39$ in testnet and $\rho = 68.23$ in mainnet. According to Fig. 6-9, it can be observed that two transactions simultaneously initiated through the same RPC provider have a higher probability to have a small interval than normal transactions across the entire Ethereum network.

Based on Equations (8)-(10), we theoretically calculate the probability of Case 1—false identification, Case 2—accurate identification and Case 3—non-unique identification per different time window T. As shown in Fig. 11, the accuracy rate (Case 2) in testnet can reach about 58.14% with a time window of 50 ms. Similarly, the theoretical accuracy in mainnet is 43.87%. Using a larger time window can have lower false identification rate (Case 1), but also lead to lower accuracy rate (Case 2) and higher non-unique identification rate (Case 3). This also demonstrates the need of the three optimization strategies in Sec. V-D.

VII. Evaluation

In this section, we present the real-world deanonymization attack results in both Ethereum testnet and mainnet.

A. Setup

We use a *Vultr* cloud sever with Ubuntu 22.04 as the probe node. It runs *DevP2P* [19] to build 25 connections to other nodes and receive transactions. We modify the source codes

of *Ethereum Wire Protocol* module in *DevP2P* to enable the probe node to only receive transactions but not relay them.

We set up a *Cisco 2960* network switch as the gateway. A victim user interacts with an RPC provider through this switch, and runs *Web3.py* library and uses *HTTPS* protocol to initiate transactions. An attacker uses a computer with Windows 11 and 16 GB memory, which connects to *Internet* through the same switch. The attacker's computer also connects to a dedicated port of the switch for port mirroring, through which the attacker's computer can capture copies of network data going through this switch. The attacker runs *pyshark* library to collect TCP packets going through the switch, uses Python to implement victim TCP packets identification rules in Sec. IV-B, and uses web3.py library to initiate a normal *Ether* transfer transaction as a beacon transaction.

B. Deanonymization Results in Testnet

The victim user initiates 1000 transactions in total to *Goerli* through *Infura*. The average time it takes for an attacker to identify victim TCP packets is about 889.16 ms.

We first evaluate the deanonymization results when the attacker uses the basic attack method in Sec. V-B, in terms of false identification rate (Case 1), accurate identification rate (Case 2) and non-unique identification rate (Case 3). As shown in Fig. 12, the real-world evaluation results are similar with our theoretical results in Sec. VI-C, which validates our models. The largest accuracy rate (Case 2) in practice is about 39.80%.

We further evaluate the effectiveness of the optimization strategies discussed in Sec. V-D, aimed at increasing the accuracy. As shown in Fig. 13, utilizing *Smallest Interval* and *Raw Transaction Size* can considerably improve the accuracy. Particularly, with a time window larger than 200 ms, the accuracy exceeds 92.80%, and can reach up to 95.40%.

Regarding the strategy of *Multiple Confirmations*, we calculate the average number of victim transactions it requires to filter out the unique result and the corresponding accuracy. As shown in Fig. 14, an accuracy of about 98.70% can be achieved with observing about only 2 victim transactions from a certain IP address.

C. Deanonymization Results in Mainnet

The victim user initiates 30 transactions to the mainnet through *Infura*, and the attacker accordingly initiates 30 beacon transactions.

As shown in Fig. 15, the practical results of the three cases in Sec. V-C have similar trend with the theoretical results with some degree of bias. Considering we only do 30 times measurements and attacks in the mainnet, the bias between the theoretical results and practical results is reasonable. The largest accuracy rate (Case 2) in practice is about 20.00%.

We further evaluate the optimized accuracy. By using the *Smallest Interval* and *Raw Transaction Size*, with a time window larger than 200 ms, the accuracy can exceed 86.67%, and can reach up to 90.00% as shown in Fig. 16.

As shown in Fig. 17, by only using the multiple confirmations strategy, the accuracy can reach about 79.31%. We carefully analyse the inaccurate cases, and find that most of them are caused by a highly active account, which frequently sends transactions. By further combining with the raw transaction size feature to filter false transactions, the accuracy can reach about 96.60% with about only 2 times of confirmations.

D. Deanonymization Results against Different RPC Providers

We compare the deanonymization results across different RPC providers when the time window is 200 ms and the smallest interval and raw transaction size strategies are adopted in testnet. The accuracy is 92.80%, 92.70%, 85.10% respectively in *Infura*, *Alchemy* and *OnFinality*, demonstrating that our attack is effective against different RPC providers.

E. Attack Cost Discussion

Our attack is of low cost, as it only requires transaction fees for initiating one beacon transaction per attack. The beacon transaction can be an *Ether* transfer transaction, whose fee is usually within 2 dollars. In the case of multiple confirmations, two times of attacks are sufficient to achieve high accuracy.

In addition, we also can exploit the deanonymization results in testnet to obtain real-world identities of users in mainnet, considering there is no charge in testnet. The insight is that a user may use the same Ethereum address to access both the testnet and mainnet. We analyse 5675 blocks in *Goerli* within a 24-hour period on July 1st, 2023. 113,869 Ethereum addresses appear in these *Goerli* blocks, and 43542 addresses out of them (about 38.24%) also active in the mainnet. These 43542 addresses hold about 2155.03 Ether balance and involve about 3,270,950 transactions in mainnet.

VIII. Discussion

We discuss two potential countermeasures that can defend against the proposed deanonymization attack. First, a user can access the third-party RPC service through a virtual private network (VPN) to hide their real IP address and improve their anonymity. Second, the Ethereum RPC protocol should be modified to add padding bytes to request and response JSON objects to ensure they are of the same size. This would make it difficult for attackers to identify victim TCP packets for sending transactions through a third-party RPC service.

IX. Related Work

In this section, we review existing work on blockchain deanonymization attacks, and attacks exploiting RPC services.

A. Clustering Blockchain Addresses

Some work clusters blockchain addresses belonging to the same user, but does not identify the real-world identity of the user. In Bitcoin, the UTXO account model [4] and the interaction pattern between layer 1 an layer 2 [20] can be used to cluster addresses. In Ethereum, features of smart contracts can be used to cluster addresses that belong to the same developer [6], [8]. Node embedding techniques can profile users based on their activity, transaction fees, and graph features [7]. In contrast, our method can uniquely determine the real-world identity of an Ethereum user.

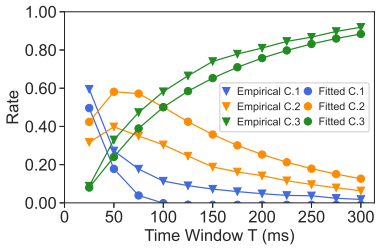

Fig. 12. Identification Results in Goerli Using Time Window T. "C." represents "Case"

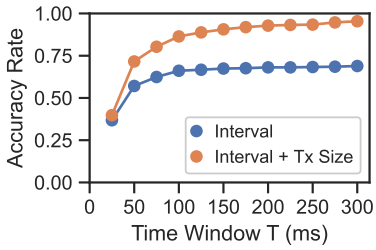

Fig. 13. Optimized Identification Results in Goerli Considering Smallest Interval and Transaction Size

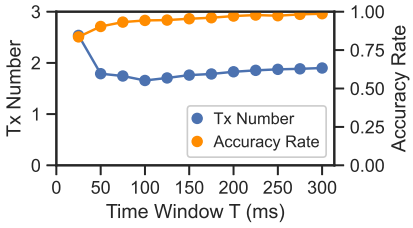

Fig. 14. Optimized Identification Results in Goerli Considering Multiple Confirmations

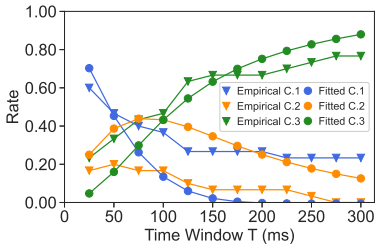

Fig. 15. Identification Results in Mainnet Using Time Window T. "C." represents "Case"

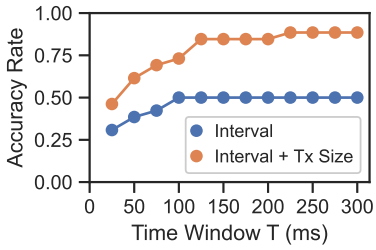

Fig. 16. Optimized Results in Mainnet Considering Smallest Interval and Transaction Size

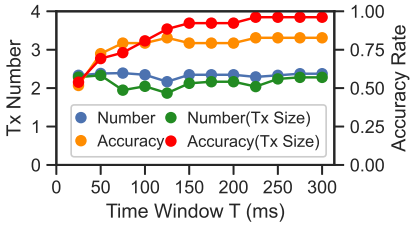

Fig. 17. Optimized Identification Results in Mainnet Considering Multiple Confirmations

B. Identifying Transaction Origin Node

Some work attempts to identify the IP address of an origin node that first propagates a transaction to the blockchain network. Biryukov, et al [9], [10] deploys a super node to connect to a large amount of nodes, so as to receive transaction from almost all blockchain nodes. The node which first sends a transaction to the super node is inferred as the origin node. Gao, et al [11] exploit the transaction propagation sequence to train a machine learning model and identify the origin node of a transaction. It also require a super node. These methods are not suitable for deanonymization in the third-party RPC service scenario. They only can correlate an Ethereum address with the IP address of a node that belongs to the RPC provider, not the IP address of a victim user. In contrast, our attack can link a victim user's IP address to an Ethereum address, and requires only one probe node in Ethereum and a few connections to other nodes (25 connections in our experiment).

C. Identifying Real-World Identities of Blockchain Users

A few existing work tries to correlate a user's blockchain address to a real-world identity such as an IP address. Biryukov, et al [15], [21] identify the IP address of a Bitcoin lightweight client/user using fixed entry nodes. However, this method cannot work in Ethereum because a Ethereum client has no fixed entry nodes [20]. A blog post [22] discusses that RPC providers collect users' IP addresses and Ethereum addresses, which violates anonymity. However, our thread model does not assume that the attacker has privileges like an RPC provider. We demonstrate that an attacker who can only monitor network data from a gateway, that a victim user connects to, can successfully deanonymize Ethereum users, under the assumption of benign RPC providers.

D. Attacks Exploiting RPC Services

A few work focuses the security problems exploiting RPC services, such as DoS attacks [23], [24], currency stealing attack [25], passphrase-extraction attack [26], behavior analysis of malicious users [27] and so on. We are the first to exploit the third-party RPC service to deanonymize Ethereum users.

X. Conclusion

In this paper, we present a novel deanonymization attack, which can accurately correlate an Ethereum address with an IP address of a user, who accesses Ethereum through a third-party RPC service. By exploiting features of TCP packets and distributions of transaction inter-arrival time intervals, an attacker can identify a victim transaction that is generated by encrypted TCP traffic from a victim user. We model the attack statistically, perform empirical measurements for modeling, and conduct real-world experiments to validate the effectiveness of our attack. With three optimized strategies, the attacker can uniquely identify the victim transaction with an accuracy of over 96% in both Ethereum testnet and mainnet.

Acknowledgment

This research was supported in part by National Key R&D Program of China (No. 2023YFC3605804), National Natural Science Foundation of China (No. 62072103), Jiangsu Provincial Key R&D Programs (Nos. BE2021729, BE2022680, BE2022065-5), Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under grants 93K-9, Collaborative Innovation Center of Novel Software Technology and Industrialization, and US National Science Foundation (NSF) Awards 2325451, 1931871 and 1915780. Any opinions, findings, conclusions, and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] Infura, "What's ahead for infura in 2022?" 2022. [Online]. Available: https://blog.infura.io/post/whats-ahead-for-infura-in-2022

[2] ——, "How infura helps metamask scale at the speed of web3 growth," 2022. [Online]. Available: https://www.infura.io/use-cases/developer-stories/metamask

[3] MetaMask, "Foxtagger snap: Mapping addresses with user-defined tags," 2023. [Online]. Available: https://metamask.io/news/developers/foxtagger-snap-mapping-addresses-with-user-defined-tags/

[4] M. Spagnuolo, F. Maggi, and S. Zanero, "Bitiodine: Extracting intelligence from the bitcoin network," in *Financial Cryptography and Data Security: 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers 18*. Springer, 2014, pp. 457–468.

[5] M. Romiti, F. Victor, P. Moreno-Sanchez, P. S. Nordholt, B. Haslhofer, and M. Maffei, "Cross-layer deanonymization methods in the lightning protocol," in *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part I*. Springer, 2021, pp. 187–204.

[6] S. Linoy, N. Stakhanova, and A. Matyukhina, "Exploring ethereum's blockchain anonymity using smart contract code attribution," in *2019 15th International Conference on Network and Service Management (CNSM)*. IEEE, 2019, pp. 1–9.

[7] F. Béres, I. A. Seres, A. A. Benczúr, and M. Quintyne-Collins, "Blockchain is watching you: Profiling and deanonymizing ethereum users," in *2021 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. IEEE, 2021, pp. 69–78.

[8] T. Chen, Z. Li, Y. Zhu, J. Chen, X. Luo, J. C.-S. Lui, X. Lin, and X. Zhang, "Understanding ethereum via graph analysis," *ACM Transactions on Internet Technology (TOIT)*, vol. 20, no. 2, pp. 1–32, 2020.

[9] A. Biryukov and S. Tikhomirov, "Deanonymization and linkability of cryptocurrency transactions based on network analysis," in *2019 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 2019, pp. 172–184.

[10] ——, "Transaction clustering using network traffic analysis for bitcoin and derived blockchains," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2019, pp. 204–209.

[11] Y. Gao, J. Shi, X. Wang, R. Shi, Z. Yin, and Y. Yang, "Practical deanonymization attack in ethereum based on p2p network analysis," in *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, 2021, pp. 1402–1409.

[12] Z. Ling, J. Luo, D. Xu, M. Yang, and X. Fu, "Novel and practical sdn-based traceback technique for malicious traffic over anonymous networks," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1180–1188.

[13] M. Shen, K. Ji, Z. Gao, Q. Li, L. Zhu, and K. Xu, "Subverting website fingerprinting defenses with robust traffic representation," in *32nd USENIX Security Symposium (USENIX Security)*, 2023.

[14] M. Tran, I. Choi, G. J. Moon, A. V. Vu, and M. S. Kang, "A stealthier partitioning attack against bitcoin peer-to-peer network," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 894–909.

[15] A. Biryukov, D. Khovratovich, and I. Pustogarov, "Deanonymisation of clients in bitcoin p2p network," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 2014, pp. 15–29.

[16] Infura, "Json-rpc methods," 2023. [Online]. Available: https://docs.infura.io/networks/ethereum/json-rpc-methods

[17] M. Finsterbusch, C. Richter, E. Rocha, J.-A. Muller, and K. Hanssgen, "A survey of payload-based traffic classification approaches," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 1135–1156, 2013.

[18] T. Wang, C. Zhao, Q. Yang, S. Zhang, and S. C. Liew, "Ethna: Analyzing the underlying peer-to-peer network of ethereum blockchain," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 3, pp. 2131–2146, 2021.

[19] M. Shen, J. Zhang, L. Zhu, K. Xu, and X. Du, "Accurate decentralized application identification via encrypted traffic analysis using graph neural networks," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2367–2380, 2021.

[20] R. Klusman and T. Dijkhuizen, "Deanonymisation in ethereum using existing methods for bitcoin," 2018.

[21] A. Biryukov and I. Pustogarov, "Bitcoin over tor isn't a good idea," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 122–134.

[22] D. Kumar, "Tips to stay anonymous while using metamask; how secure is metamask?" 2022. [Online]. Available: https://coingape.com/blog/how-to-stay-anonymous-while-using-metamask/

[23] 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.

[24] K. Li, Y. Wang, and Y. Tang, "Deter: Denial of ethereum txpool services," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 1645–1667.

[25] 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 *RAID*, vol. 2019, 2019, pp. 47–60.

[26] X. Wang, X. Zha, G. Yu, W. Ni, R. P. Liu, Y. J. Guo, X. Niu, and K. Zheng, "Attack and defence of ethereum remote apis," in *2018 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2018, pp. 1–6.

[27] K. Hara, T. Sato, M. Imamura, and K. Omote, "Profiling of malicious users targeting ethereum's rpc port using simple honeypots," in *2020 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2020, pp. 1–8.