Skylar Scanner - Navigating the ERC-4337 Standard with a Custom Explorer

https://skylar-scanner.vercel.app/

Hans Bhatia, Winston Yeo, Vikram N. Subramanian
David R. Cheriton School of Computer Science
University of Waterloo
{h23bhati, khwyeo, vnsubram}@uwaterloo.ca

***Abstract*—This paper explores ERC-4337, a new Ethereum standard that introduces account abstraction. ERC-4337 enables improved user experiences in cryptocurrency transactions and wallet management. This paper discusses the architecture of ERC-4337 and discusses the challenges and opportunities ERC-4337 brings to the Ethereum ecosystem. We also introduce Skylar Scanner, a transaction explorer designed to navigate and interpret 4337 transactions. The tool empowers both developers and users by offering easy access to transaction details, balances, NFTs, and other essential information. We also conducted a developer survey to gather insights on ERC-4337, identifying opportunities and hurdles associated with its adoption. The findings emphasize the need for improved documentation, tooling, and developer communities to maximize the potential of ERC-4337 and foster its integration into the broader Ethereum landscape.**

I. Introduction

ERC-4337, also referred to as account abstraction, is an Ethereum standard that considers using smart contracts as wallets instead of externally owned accounts without requiring changes to the consensus layer. It was proposed by Ethereum co-founder Vitalik Buterin and other developers in 2021. The specification specifies a set of contracts that contain various functions to aid in the emulation of transaction submission, transaction payment (gas fees), signature verification and transaction execution. This innovation enables more user-friendly crypto wallet designs, making tasks like transaction signing, transaction verification (eg. multisig), and other wallet functions more convenient and flexible. (1)

This standard is a significant step toward improving the user experience and accelerating the adoption of cryptocurrencies. As the specification proposes programmable wallets, wallets can be enhanced with features like account recovery, one-click transactions, and more. With the first stable set of contracts implementing ERC-4337 being deployed to the Ethereum mainnet in March 2023, ERC-4337 has the potential to transform the way users interact with crypto and could lead to the development of innovative and user-friendly wallet designs in the future. (2)

Externally Owned Accounts (EOA) - are addresses identified by the public key of an elliptic curve (3) generated private key and its corresponding public key. EOAs are stored on the Ethereum ledger as a 42-character hexadecimal address derived from the last 20 bytes of the public key with their corresponding native coin balances. The private key is used to sign transactions that can be verified through cryptographic operations. These signed transactions are submitted to the blockchain and form the basis of an EOA engaging with smart contracts. Since transactions cost native coin, the blockchain nodes can easily look up their balances.

Smart contract accounts (SCA) - are contracts within the Ethereum blockchain that emulate the behavior of an EOA by containing two parts. First, having a way to accept or reject transactions as coming from a given SCA address. Second, having a way for the SCA address to look like it is the one carrying out the transaction.

SCA are not controlled by traditional private keys like EOAs but are governed by the logic programmed into the contracts. The start of any transaction begins from an EOA which submits a transaction to the SCA. The SCA can then verify the request and determine whether or not to approve it. Once approved, the SCA takes advantage of the call function in the Ethereum Virtual Machine to process the actual transaction. This function will make it such that in other contracts, the address making the transaction now belongs to the SCA over the EOA. In particular, when other contracts checks for the sender via the msg.sender parameter in the solidity programming language, it would reflect the SCA rather than the EOA.

Bundlers- Without an EOA, however, a transaction cannot be submitted. To combat this, ERC-4337 introduced an alternate mempool where users can submit pre-signed operations (UserOps) too. A new group of providers, called bundlers can then run batch these requests up into a bundle and use their own EOA to submit the bundle onto the chain. As a form on incentive, bundlers often get some fee paid out after the successful execution of each bundle on the chain.

Transaction Explorers- are used in order to view the history/details of a transaction, block, smart contract, or EOA. They provide transparency, verifiability, and debugging user-interfaces; significant for developers and users.

With an entirely new wallet architecture consisting of different transaction hashes introduced by ERC-4337, there is a lack of a good transaction explorer for ERC4337 transactions.

This paper aims to fix that by introducing Skylar scanner.

The key ERC-4337 infrastructure consists of multiple parts-

1) **EntryPoint** - a singleton contract to execute bundles of UserOperations. Bundlers/Clients whitelist the supported entrypoint.
2) **Bundler** - The new group of providers that takes in userOperations and create a valid an EntryPoint.handleOps() transaction. They would then attempt to add it to the block while it is still valid. This can be achieved by a number of ways:
   a) Bundler can act as a block builder itself
   b) If the bundler is not a block builder, it MUST work with the block building infrastructure such as mev-boost or other kind of PBS (proposer-builder separation)
   c) The bundler can also rely on an experimental eth_sendRawTransactionConditional RPC API if it is available.
3) **Sender** - the SCA that is sending a user operation.
4) **UserOperation (userOp)**- This describes the transaction object that consisting of the wallet metadata, calldata and signature to be verified by the verify function implemented by the SCA.
   The simplified flow from userOp creation to execution goes as such:
   a) The user with a deployed SCA creates a userOp object, attaching data for authentication of the operation (e.g a signature), transaction calldata, transaction metadata, and gas limits.
   b) The user sends the object to a new alternate mempool for userOps.
   c) a bundler picks up on this userOps, and bundles it into a bundle that they push on-chain via the Entry point contract.
   d) The Entry Point contract interacts with the SCA, calling the verify function, and the execute function. Upon successful interaction, it also compensates the bundler. All these happen atomically.

   The UserOp is used to describe what transaction a user wants to execute to the smart contract wallet. The UserOp looks like the following object

```
userOp {
    sender,
    nonce,
    initCode,
    callData,
    callGasLimit,
    verificationGasLimit,
    preVerificationGas,
    maxFeePerGas,
    maxPriorityFeePerGas,
    paymasterAndData,
}
```

   While the actual values are not important, the details behind the derivation of the userOp hash is. To get the userOp hash associated with the above userOp, we first concatenate all the values above. Once we concatenate all the above values, we then keccak hash (4) the concatenated values and concatenate the hashed value with the entry point contract address and the chain id that the userOp is being submitted to.
   The final value is then keccak hashed one last time to obtain the userOp hash. (5). This userOp hash will uniquely allow us to identify the userOp.

II. Motivation

The goal of account abstraction is to allow users to use smart contract wallets containing arbitrary verification logic instead of EOAs as their primary account. As such, it removes the need for users to have EOAs. This is in contrast to the current smart contract wallets, and other similar proposals like EIP 3074 (6).

Alongside the goal of a better account model on the blockchain, ERC-4337 seeks to maintain two important properties - decentralization and not requiring any consensus changes. The former is important as it aligns with the core value of Ethereum today, and helps avoid trusting any bundlers. In particular, anyone is allowed to be a bundler, and users are free to submit their transactions to any bundler to be included. On the other hand, the latter is important in ensuring adoption. Various other proposals have been made aside from ERC-4337 attempting to introduce account abstraction. This included EIP 3074 and EIP 5003. (7) However, both of these proposals required changes to the underlying protocol. With the Ethereum consensus layer focusing on scalability-oriented features, attempting to introduce change at that level is unlikely to happen (1).

While the proposal has managed to address these two issues, it failed to consider the wider impact it would have on the ecosystem of decentralized applications (dApps) and current users today. In particular, the lack of support for smart contract wallets by dApps and the lack of transparency into userOp for users are some of the reasons modern adoption is hindered today. We discuss some of these issues in detail below.

A. Not recognized by various decentralized applications (dApps)

Due to the recency and complexity, the specification is not compatible with today's EIP-1193 (8). In particular, EIP-1193 introduces a universal way for developers of dApps to connect to the user's wallet and send transactions. The send transaction specification, however, expects a transaction hash of the submitted transaction as a return value. dApp developers then use this transaction hash value to check on-chain and report the status accordingly. However, the transaction hash returned by ERC-4337 is derived and does not actually correspond to an actual on-chain transaction's hash. As a result, many applications end up throwing errors when the userOp's hash is returned. This issue is significant

as it makes existing dApps feel broken and unusable, the opposite of what the ERC-4337 specification hopes to solve.

For example, one of the top DEXs, Uniswap, does not detect the success of a UserOp, returning a failure even in the event of a successful userOp. Hence, users are always greeted with an error screen when interacting with Uniswap regardless of the outcome of their transactions. This provides a clunky experience.

B. No way of accessing UserOp transactions

The major Blockchain explorers do not provide this feature since it is distinct from regular transactions and relatively new. As a result, users cannot view their UserOps by simply looking up its hashes on these explorers. Without knowledge of the specification and the Blockchain, it would arguably be impossible for the user to find the transaction details.

On the other hand, Developer's can find these transactions by writing small scripts or inventing a custom library to parse through the transactions. However, this is time-consuming, complicated, and adds bloat. We outline one such method of deriving these transaction hashes in our implementation later. To expect every dApp developer to put in extra work to parse these userOp transactions within their already complicated dApp is unlikely to happen.

C. Lack of debugging infrastructure for developers

The ERC-4337 specification defines a list of error codes that must be returned in different scenarios. A compliant architecture will pass a test suite that re-creates the specific scenario in which the expected codes are thrown. The defined error codes, however, are quite general. This makes it hard for developers to properly identify issues within their implementation and/or integration of the ERC-4337 specification.

Bringing everything together, we see that while ERC-4337 solves some problems around private key management and makes adoption by the protocol easy, there is still some work to be done in terms of developer adoption. In particular, the EVM transaction explorer that we build seeks to address the second issue of making it easy to access the userOp transaction.

III. Technical Architecture

This section of the paper details how the EVM transaction explorer was built to parse userOp transactions alongside regular transactions. In order to understand parsing and querying, we will first cover how transactions in EVM are indexed and managed, followed by how logs within these transactions are indexed and managed.

We will then go over how our system leverages these workings of the EVM to parse and filter userOp as quickly as possible.

A. Transactions

Transactions form the bedrock of the EVM. It allows any externally owned account (EOA) to interact with EVM and make changes to the global state. (9)

Every transaction has a transaction hash. This hash is calculated based on the various fields of the transaction listed below:

```
Transaction {
    nonce
    gasPrice
    gasLimit
    to
    value
    data
    ecdsaV
    ecdsaR
    ecdsaS
}
```

While the actual correspondence of the values above is not important, know that these values are essentially concatenated and signed with the private key of the user submitting the transaction. The resulting signed transaction is passed to the keccak hash function which will return us with the final hash associated with the transaction. (10)

For every block that gets proposed, it contains a list of transactions in the block's body. The transaction hashes of these transactions are then stored and indexed by the many Ethereum clients such as Geth. In particular, the Geth client creates an index of transaction hashes to the block number that contains it along with the transaction index. (footnote here) This allows for a constant look-up time for any given transaction hash.

Now it might be important to distinguish between the various types of clients. There are full clients and light clients. The former downloads and indexes all data from the chain, while the latter only download block headers. Block headers do not contain the list of transactions but rather only a transaction root. The transaction root is a value corresponding to the Merkle root of the list of transaction hash (footnote).

Hence while full nodes can query for transaction hash in $O(1)$ time on their own, light clients rely on full clients to tell it if the transaction exists or not. Light clients also rely on the full clients to provide the proof so that it can verify that the full client is indeed telling the truth (footnote).

Given that we now know how transactions are queried by various clients on the EVM, we turn to how logs within the transaction are queried.

B. Logs

Logs are not directly indexed even by the full clients. Instead, bloom filters are leveraged. A Bloom filter is a data structure designed to allow rapid and memory-efficient identification of whether an element is present in a set.

The downside is that a bloom filter is a probabilistic data structure. It is used to determine if either the log is definitely not in the set or that it may be in the set. (11)

The base data structure of a Bloom filter is a Bit Vector, and the Ethereum virtual machine uses a log bloom of 256 bytes or 2048 bits. (12)

The logs bloom is stored on two different levels, the block, and the transaction. This will become apparent in its usefulness when we go over how userOp is parsed.

C. Parsing for userOp and regular transactions

A userOp transaction in the EVM is essentially ephemeral. Since they are basically function calls to the $handleOps$ function on the entry point contract, there is no way to track them down via an indexed look-up against any full nodes. That said, every userOp transaction has a hash derived from the parameters of the $handleOps$ function. This hash is what the user is given as part of the proof that their transaction made it onto the blockchain.

The userOp hash looks exactly the same as a regular transaction hash. In particular, both correspond to some keccak hash of an object. As such, there is no obvious way to determine if the hash given is a regular transaction or not.

Hence we first leverage the way transaction look-ups can occur via full nodes in constant time. This allows us cheaply know whether a transaction is a proper EVM transaction or not quickly

If a transaction is a proper EVM transaction, we are done. Otherwise, we can leverage the bloom filters to parse the appropriate logs quickly. This is done by first checking the log bloom on the block for a hit. We only dive and check the specific transaction log bloom only if the block's log bloom suggests a match. Finally, when we get a match on the transaction's log bloom, we then slowly parse through each and every log within the transaction checking for the specific log that is emitted. In practice, this has worked extremely well and has allowed us to parse userOp logs in low single-digit seconds.

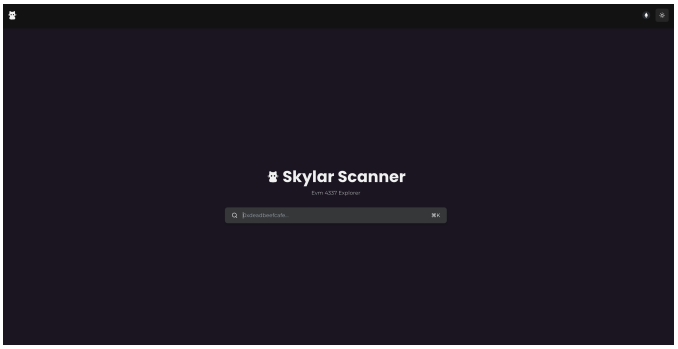

Fig. 1. Homepage

IV. Tool usage

We created Skylar Scanner, a transaction explorer that makes it easy for both users and developers to interact with the overall 4337 transaction infrastructure (bundle txns, spec-related contracts, user ops). This section will highlight the features of the tool.

Skylar scanner is live and can be accessed at https://skylar-scanner.vercel.app/.

The homepage in figure 1 welcomes the user with a search bar that supports queries user op hashes. Additional query features for regular transactions, EOA's, and smart contracts were added so that the tool is on par with the many other Blockchain explorers. The tool supports Ethereum, Polygon, and their respective testnets for the contracts deployed by the Ethereum Foundation.

We've also added shortcuts for developers to easily focus on the search bar, change networks, copy hashes, and input data. With a simple search, our tool classifies the type of query and then retrieves the data if valid.

In figure 2, the page for a UserOp hash query result is shown. The important UserOp data fields are shown and all costs are converted to dollars. Here a developer/user can easily find important details regarding their transaction. Apart from the accounts page, the other queries result in similar-looking pages.

In figure 3, the page for an EOA or SCA can be seen, here we display user's balances (tokens + native currency), NFTs, and transactions.

V. Developer Survey

While the tool we built solves a small problem in the adoption of the ERC-4337 specification today, we wanted to better understand the potential of ERC-4337. As such, we surveyed developers regarding the opportunities they perceive with ERC-4337 and the challenges they encounter when working with this standard. Our goal in documenting developer perspectives is to contribute a clearer path to the advancement of ERC-4337's adoption, development tooling, and overall impact on the Ethereum ecosystem.

A. Survey Method

We conducted a qualitative survey via semi-formal interviews with crypto developers on EVM today. The interviews were either in-person or through video calls. The framework followed is presented below.

To get interviewees, we contacted friends and professional contacts who had prior experience working with Web3 and specifically on EVM. We ensured they had a reasonable amount of experience in the area. If they were not aware of ERC 4337, we encouraged them to read up on it before the interview. We also provided a brief overview of ERC-4337 prior to conducting the interview. We had 10 interviewees in total,

B. Demographic Information

1) What is your current role/job title?
2) How many years of experience do you have in blockchain and smart contract development?
3) Have you previously worked with Ethereum standards like ERC-20 or ERC-721?

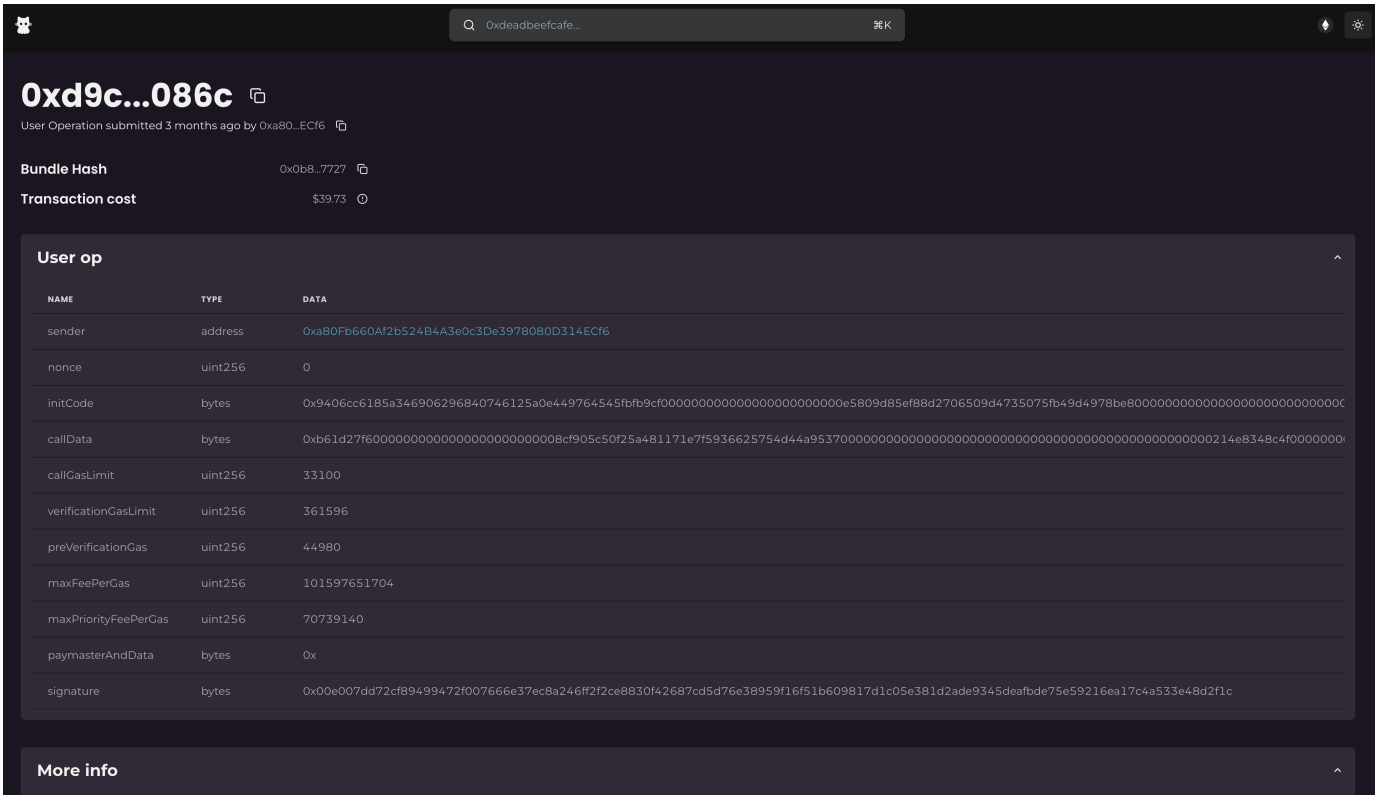

Fig. 2. UserOp Page

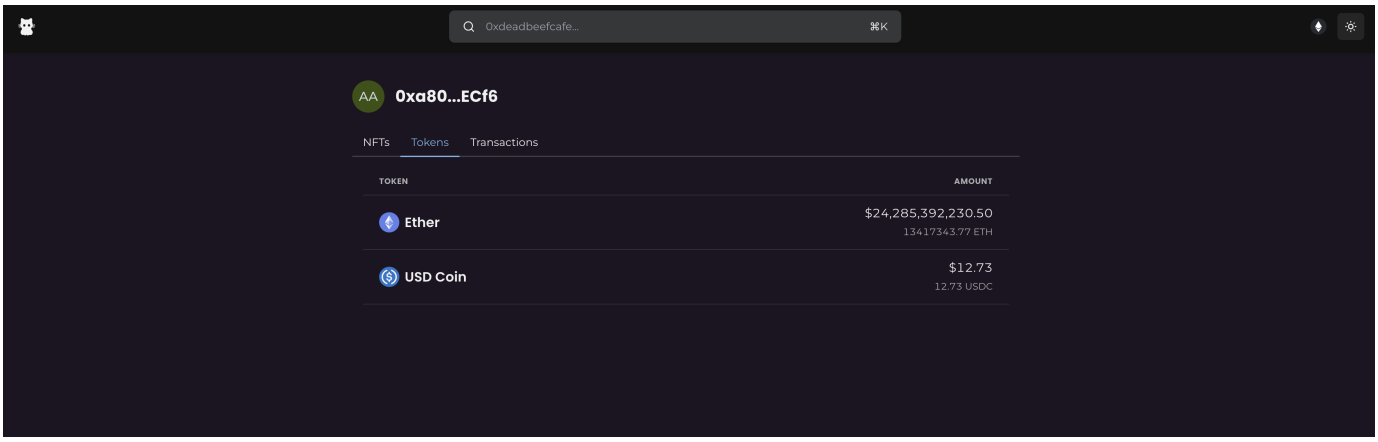

Fig. 3. Account page

C. Opportunities with ERC-4337

1) In your opinion, what are the key opportunities that ERC-4337 presents for the Ethereum ecosystem?
2) How do you envision ERC-4337 contributing to the development of user-friendly crypto wallets?
3) Can you suggest potential innovative use cases enabled by ERC-4337's account abstraction? (e.g., aggregated signatures, daily transaction limits, emergency account freezing, privacy-preserving applications)

D. Challenges and Hindrances

1) Have you encountered any challenges or hurdles when working with ERC-4337?
2) Are there any specific technical challenges you've faced while developing applications using ERC-4337? (e.g., lack of development tools, documentation)
3) How do you perceive the availability of tooling, libraries, and resources for ERC-4337 development? Is there a need for additional support?
4) What are the main obstacles you see in the adoption and

implementation of ERC-4337 for wider use?

E. Suggestions and Feedback

1) Based on your experience, what improvements or enhancements would you suggest for ERC-4337 or its associated tooling?
2) Have you used our tool, Skylar scanner? What do you think of it? Any good/bad things about it?

F. Open Comments

1) Is there anything else you would like to share about your experience, insights, or concerns related to ERC-4337?

VI. Survey Results

In this section, we present the findings and insights obtained from the survey conducted with crypto developers to gauge their perspectives on ERC-4337.

A. Demographic Information

The survey was conducted with a total of 10 crypto developers who had a background in blockchain and smart contract development. The participants were a mix of students and professionals from diverse roles within the industry including startup founders in the Web3 space and more experienced developers.

The respondents had a median of 2 years and a mean of 3.4 years of experience in blockchains. 8 participants had prior experience working with Ethereum standards like ERC-20 or ERC-721.

B. Opportunities with ERC-4337

The key opportunities perceived for the Ethereum ecosystem through ERC-4337 included:

1) Respondents envision ERC-4337 contributing to user-friendly crypto wallet development by simplifying account management, allowing multi-factor authentication, and enabling more intuitive transaction processes.
2) Setting daily transaction limits to enhance security and prevent overspending.
3) Emergency account freezing to mitigate security threats.
4) Privacy-preserving applications that utilize a single account for discreet transactions.
5) Fine-grained atomic batch-transactions leading to applications like multi-swaps.
6) Token gas payments - alleviating the need to hold + pay with the native token.

C. Challenges and Hindrances

While most respondents expressed positive views about ERC-4337, some encountered challenges such as the need for comprehensive documentation and resources for implementation.

1) Limited availability of development tools and libraries specific to ERC-4337.
2) The availability of resources for ERC-4337 development was perceived as somewhat limited, with a consensus that additional support and educational materials would greatly benefit the developer community.
3) The learning curve is associated with understanding the intricacies of account abstraction.
4) Integrating ERC-4337 into existing projects and transitioning from conventional account models.

D. Suggestions, Feedback and Open Comments

1) Enhancing documentation with real-world use cases and code examples.
2) Respondents expressed a desire for comprehensive tutorials, sample projects, and a dedicated developer community to foster a better understanding of ERC-4337.

E. Feedback about our tool

1) The home page is uncluttered unlike majority of Blockchain explorers which have analytics that are generally not useful.
2) The tool allows for quickly searching the contents of a transaction with the shortcuts, at the same time the labels are very intuitive even for a developer used to the jargon.
3) A developer in the 4337-Bundler space expressed a desire to incorporate error codes into the explorer as it can be quite cumbersome to figure out why a transaction failed.

VII. Future Directions

In this section, we present some avenues through which we can improve Skylar Scanner.

A. Efficient Logging

Searching by UserOp hashes required diving through the Log Bloom filter. In order to make this infrastructure better, without having to add anymore overhead to the Ethereum network, it will be beneficial to research better lookup mechanisms. One possibility would be to host a public database of all user operations while constantly listening for new UserOp events for the entry point contract desired, from here the explorer can simply optimize the database schema and get faster lookups without redundant node usage.

B. Debugging Infrastructure - Error Codes

Blockchain companies frequently waste resources on building the same tooling over and over, whereas a good explorer would be able to cover most of the required tooling that developers would need.

1) Error Codes - As the feedback we received pointed out, there should be a way to view error codes for failed transactions. It would be extremely useful to look at supporting and displaying error codes required by the spec and possibly even add more fine-grained error codes when simulating a transaction.
2) Interactability - For developer workflows in order to quickly test out inputs it would be useful to provide a lightweight tool that executes transactions on a local forked node.

VIII. Conclusion

In conclusion, this paper covered the ERC-4337 specification. Starting with an overview of the specification itself, we then went into issues hindering adoption by the general developer community despite much enthusiasm from the protocol and Ethereum foundation. Following which, we introduce Skylar Scanner, an EVM transaction explorer that also parses userOp transactions alongside regular transactions. Finally, we presented a survey of 10 blockchain developers going over where they think some of the biggest hurdles today in the adoption of ERC-4337 alongside some of the opportunities. We conclude with some future directions that could be taken with Skylar Scanner to help it continue to address some of the pain points for developers.

References

[1] V. Buterin. (2021) Erc-4337: Account abstraction using alt mempool. [Online]. Available: https://eips.ethereum.org/EIPS/eip-4337

[2] B. Academy. (2023) What is erc-4337 or account abstraction for ethereum. 2023. [Online]. Available: https://academy.binance.com/en/articles/what-is-erc-4337-or-account-abstraction-for-ethereum

[3] D. Hankerson and A. Menezes, "Elliptic curve cryptography," in *Encyclopedia of Cryptography, Security and Privacy*. Springer, 2021, pp. 1–2.

[4] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "Keccak," Cryptology ePrint Archive, Paper 2015/389, 2015, https://eprint.iacr.org/2015/389. [Online]. Available: https://eprint.iacr.org/2015/389

[5] StackUp. (2023) Useroperation hash. [Online]. Available: https://docs.stackup.sh/docs/erc-4337-useroperation-hash-guide

[6] S. Wilson. (2020) Eip-3074: Auth and authcall opcodes. [Online]. Available: https://eips.ethereum.org/EIPS/eip-3074

[7] D. Finlay. (2022) Eip-5003: Insert code into eoas with authusurp. [Online]. Available: https://eips.ethereum.org/EIPS/eip-5003

[8] Ethereum Improvement Proposals. (2018) Eip-1193: Ethereum provider javascript api. Accessed: August 11, 2023. [Online]. Available: https://eips.ethereum.org/EIPS/eip-1193

[9] mlibre. (2023) Transactions. [Online]. Available: https://ethereum.org/en/developers/docs/transactions/

[10] T. Værge. (2018) How to calculate the assigned txhash of a transaction? [Online]. Available: https://ethereum.stackexchange.com/questions/45648/how-to-calculate-the-assigned-txhash-of-a-transaction

[11] llimllib. (2019) Bloom filters by example. [Online]. Available: https://llimllib.github.io/bloomfilter-tutorial/

[12] E. Foundation. (2021) Ethereum logs bloom. [Online]. Available: https://ethereum.github.io/execution-specs/autoapi/ethereum/frontier/bloom/index.html