IBIMA Publishing
Communications of International Proceedings
https://ibimapublishing.com/p-articles/44AI/2024/4435124/
Vol. 2024 (22), Article ID 4435124

Exploring Account Abstraction in Ethereum: A Deep Dive into Erc-4337*

Kamil KACZYŃSKI and Aleksander WIĄCEK

Military University of Technology, Faculty of Cybernetics,

Institute of Mathematics and Cryptology, Poland

Correspondence should be addressed to: Kamil KACZYŃSKI, kamil.kaczynski@wat.edu.pl

* Presented at the 44th IBIMA International Conference, 27-28 November 2024 Granada, Spain

Abstract

Ethereum is a decentralised, censorship-resistant blockchain. It allows users to run decentralised applications and deploy Turing-complete smart contracts. However, the Ethereum blockchain is rather stagnant compared to Bitcoin and its ability to introduce new features to its chain. The Ethereum blockchain, unlike Bitcoin, has an Ethereum Virtual Machine (EVM) built into the chain and lets its users run decentralisation and deploy smart contracts, which fulfils one of the main ideas of blockchain. ERC-4337 introduces the possibility of using classic signature algorithms that are more efficient than ECDSA or multiparty computation and threshold signature schemes that further increase security. It also introduces post-quantum signature schemes in the blockchain. This change also allows the use of aggregators that can utilise Boneh-Lynn-Shacham signature schemes, which have also been used in the Ethereum beacon chain protocol. Also, making most of the in-use smart contracts deployed on the chain compatible with ERC-437 can be challenging as they must be upgraded or, if that is not possible, redeployed with code changes. In general, it can make blockchain interactions user-friendly without losing any of the major Ethereum blockchain principles. It laid the foundation of what Ethereum can become in the future, with Web2 functionality.

Keywords: Ethereum, ERC-4337, blockchain, DeFi

Introduction

Although Bitcoin, considering its market capitalisation [1], can be called one of the most popular blockchain networks worldwide, it is rather stagnant compared to Ethereum and its ability to introduce new features into its chain. The Ethereum blockchain, unlike Bitcoin, has an Ethereum Virtual Machine (EVM) built into the chain and allows its users to run decentralised applications and deploy Turing-complete smart contracts [2] [3]. A smart contract (SC) can be thought of as a self-executable program running on blockchain, designed in a specific language, for example, solidity. It consists of functions that can be called by SC users and a state that can change when users execute SC functions. A deployed smart contract has its address, visible to anyone. Smart contracts are immutable, which fulfils one of the main ideas of blockchain, but it can also sometimes be a drawback and lead to workarounds, for example, when there is a need to upgrade the smart contract. The Ethereum account is an object that allows blockchain users to send and receive ETH currency or ERC-20 tokens [4]. There are two types of accounts. The first one is an Externally Owned Account (EOA), which can be simply represented as a pair of private and public keys of the Elliptic Curve Digital Signature Algorithm. The address for this type of account comes from the last 20-byte portion of the Keccak-256 hash of the public key. To transfer funds, a sender must sign the prepared transaction using the private key. This account is managed by the user and does not allow for custom logic implementation. The second type of account is a contract account. Here, the account is essentially a smart contract and is fully managed by the SC code executed in EVM. There are no cryptographic keys involved. Unfortunately, these accounts do not initiate transactions independently. They execute them only in response to

Cite this Article as: Kamil KACZYŃSKI and Aleksander WIĄCEK, Vol. 2024 (22) "Exploring Account Abstraction in Ethereum: A Deep Dive into Erc-4337 " Communications of International Proceedings, Vol. 2024 (22), Article ID 4435124, https://doi.org/10.5171/2024.4435124

calls received from EOAs or other smart contracts. Therefore, the need for an EOA is always present. The contract account is managed by code. As we can see, the use of Ethereum accounts requires some technical knowledge. Even though there are solutions that try to be user-friendly, for example, the ledger cold wallet as EOA, it can still be hard to safely manage secret phrases for the average person. ERC-4337 is a chain proposal that introduces account abstraction [5]. ERC-4337 takes the best of both Ethereum account types and 'puts' programmable smart contract accounts into user wallets and introduces arbitrary verification logic. Here, EOA accounts are no longer needed, and every account can initiate transactions. Wallets adopting account abstraction can be more user-friendly, which can encourage people to use the Ethereum blockchain. The risk of losing money due to the loss of private keys is much lower. Public-key cryptography is still used in wallets, but there are much more reliable recovery methods, such as multi-factor authentication recovery, social recovery, and many others. In account abstraction, there is no need to rely only on ECDSA. ERC-4337 introduces the possibility of using classic signature algorithms that are more efficient than ECDSA or multiparty computation and threshold signature schemes that further increase security. This change also makes the introduction of post-quantum signature algorithms in blockchain. Users can select wallets using these algorithms to protect their accounts against future threats posed by the development of quantum computers. Note that bundlers, explained in Chapter 2 of this paper, still sign transactions with ECDSA using their private key, so ERC-4337 does not fully eliminate postquantum vulnerabilities. The other new functionality is the ability to pay fees by users in ERC-20 tokens, eliminating the need to hold ETH. Users can also have their fees paid by third parties [5].

The terms 'wallet' and 'account' should be distinguished, as they are not the same. A wallet is a user interface that allows the control of an Ethereum account, which involves creating transactions. For EOA, it manages the user's cryptographic keys [3].

ARCHITECTURE OF ERC-4337

Account abstraction does not require a fork to be adopted into the chain. It does not change the consensus layer but operates on a high-level replication of the transaction mempool. The architecture consists of a few new components. In ERC-4337, instead of the classic transaction that is propagated into blockchain nodes, wallets construct an object called UserOperation that is further encoded with ABI. Such an object is sent to a dedicated UserOperation mempool for the bundler to deal with. Before adding User Operation to the mempool, the object has to be simulated off-chain [5]. In other words, UserOp is checked to see if it is valid and covers the cost of its execution while respecting some additional rules of the alt mempool [5]. The second simulation occurs before creating the bundle. The structure of UserOperation has some similarities to the standard EIP-1559 transaction.

Fields in operation similar to this are [2]:

- **nonce** that is used to prevent replay attacks; the structure of nonce is different than in transactions, and it has a sequential value and a key that the user can change. It adds to the customizability of the object and can guarantee that the hash of UserOp is unique on the chain,
- **maxFeePerGas,** which says about the highest amount to be paid per unit of gas for the transaction [6],
- **maxPriorityFeePerGas** regarding the maximum amount of gas to be thrown as a tip to bundler, which as result prioritizes the **UserOperation** object**,**
- **calldata** used to interact with smart contracts,
- **signature,** but, in contrast to standard transactions, with the custom algorithm.

Fields that are exclusive to the UserOperation object [5]:

- **sender** with the address of an account operating,
- **factory** and **factory data** used to deploy account on the chain when performing create2 EVM opcode,
- **callGasLimit,** setting the gas limit for the execution phase,
- **verificationGasLimit** analogously for the verification phase,
- **preVerificationGas** that is calculated before the operation is verified and depends on the complexity of an object,
- **paymaster,** which contains the address of the Paymaster contract,
- **paymasterData** contains any additional paymaster data needed for verification and execution,
- **paymasterVerificationGasLimit** and **paymasterPostOpGasLimit** say how much gas to allocate for the paymaster validation and post-operation code.

Ethereum's Proof-of-Stake (PoS) consensus involves validators, which are simply special nodes that verify and propose new blocks. Bundlers are a little like validators, but instead of proposing blocks, they collate UserOperation objects creating bundle transactions. Bundlers monitor the dedicated mempool for new UserOp objects [5]. A batch with UserOps will be sent with a handleOps or handleAggregatedOps call to the global

EntryPoint contract. Those two calls have the same logic, but handleAggregatedOps is used when the bundle contains UserOperations of multiple aggregators. The role of the EntryPoint contract is very important, as it handles the creation of new accounts, verifies the signature of UserOp objects, and ensures fee payments [5]. The entry-point contract calculates the maximum possible fee and the fee that must be added to the entry-point account deposit. It also checks if the user account has enough funds to cover the maximum operating cost. The validation of UserOp is separated from its execution. EntryPoint uses the validateUserOp method to verify the signature and pay the fee [5]. The execution phase starts when EntryPoint has successfully verified all userOps in the bundle. Here EntryPoint essentially executes the logic on target contracts that was defined in the calldata field in UserOp objects. Then, it returns any unused precharged gas to account wallets and transfers the accumulated fees from all UserOperations to the address specified by the bundler. EntryPoint can support the simulation process, but its functions must be extended by a bundler with the EntryPointSimulations contract code [5]. Every wrong UserOperation object is removed from the batch. Figure 1 shows a basic high-level diagram of the ERC-4337 architecture.

EntryPoint allows for the use of the Paymaster mechanism, in which it calls paymaster contracts confirming their agreement to fund transactions. Paymasters allow users to pay fees in ERC-20 tokens and even make it possible to fully subsidise the fees. The paymaster contract receives the required value of ERC-20 tokens from the user account and pays for their fees with ETH [7]. When the paymaster is set in UserOp, then in the verification phase of handleOp, EntryPoint checks if the paymaster has enough amount of ETH in their deposit. Next, there is a call on paymaster to find out if it is willing to pay the fee, and if so, the right amount of it is taken by EntryPoint. In the execution phase, the paymaster receives feedback on the success of the execution and calculates fees in the form of another call [5].

ERC-4337 implements a reputation system for paymasters. In this system, the paymaster can either be banned, throttled, or in OK state. The state depends on how many good UserOps related to Paymaster were included in the bundle [8]. The paymaster's logic is taken into account during the simulation phase. An aggregator is another contract in ERC-4337 that improves the signature verification process. This trusted contract aggregates UserOps signatures with specified aggregator addresses into a single signature. HandleOps also runs an on-chain function that verifies such signatures against all UserOps to check if the aggregation is correct. Aggregators must also be verified and cannot be in a throttled or banned state [5].

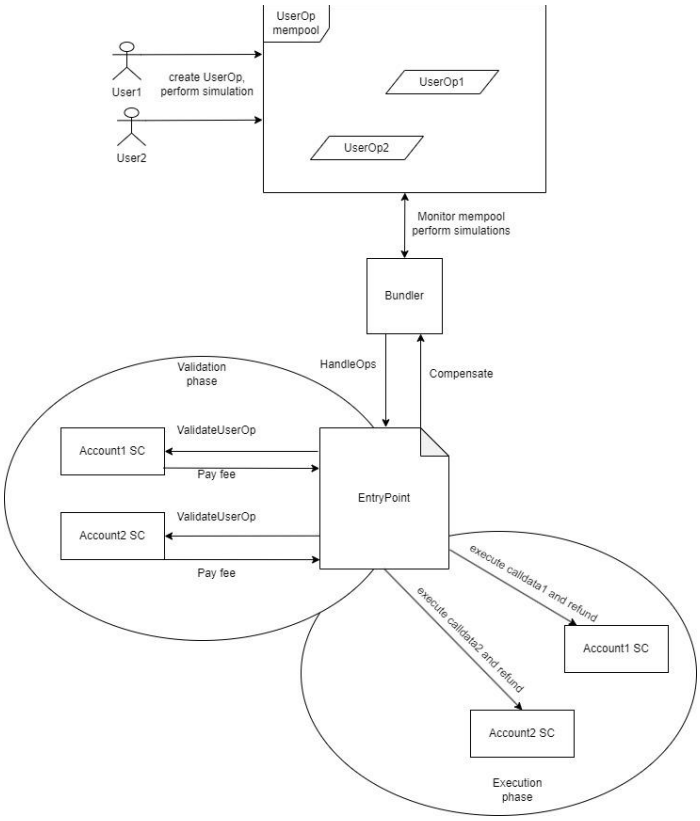

Figure 1. High-level basic architecture of ERC-4337

Impact On Dapps

Comparison with existing Ethereum transaction methods

Currently, on Ethereum, three types of transactions can be distinguished: transfer of funds between EOA accounts, those that interact with smart contracts and those that implement smart contracts [2]. The lifecycle of those transactions is mostly the same and differs in the execution phase. In proof of stake, a randomly chosen validator is obliged to create a new block and pass it on to other nodes. The block is checked if it's valid by randomly chosen validators that form a committee. For the transaction to be included in future block, it has to overcome a few steps. First, the user creates the transaction and signs it with a private key. Then, the prepared transaction is sent to the Ethereum node. Upon receiving the transaction, the execution client of the node checks if the signature of the transaction is correct and if the user has enough ETH [9]. After verification, the transaction is added to the pool where it is pending. Then, a single validator node selects transactions from mempool, bundles them into a new block, and performs local state change by executing them. This block is broadcast to other validator nodes and verified by them. The transaction can then be completed [9]. Comparing it to account abstraction, one can see that the whole process of ERC-4337 UserOp bundling happens before the transaction lifecycle begins. In such architecture, the wallet, instead of signing the transaction and sending it to the node via RPC, now creates UserOp and sends it to the bundler. Bundlers batch UserOps and put them into a new transaction that will be included in a block, which essentially can be considered the third step of the transaction lifecycle. Because there is no change to how transactions and blocks are verified, the consensus layer stays the same, and the chain does not require a hard fork.

How ERC-4337 Enhances User Experience and Security

Currently, EOA account recovery (in a deterministic wallet) depends on a seed, which essentially is a master private key used in key derivation algorithms. Losing it equals losing access to funds. While most of the time, seed can be presented as mnemonic code words, it still can be considered as not user-friendly because users must remember it or find a safe place to hide it, usually for a very long time. It is a single point of failure, where the adversary using a phishing attack needs only the seed phrase to take control of someone's wallet. As discussed earlier, ERC-4337 accounts are smart contracts that do not use private keys directly to sign transactions. User accounts are triggered by EntryPoint to execute calls, and since they are smart contracts, their logic can be implemented in such a manner that they can verify the signature of a custom scheme. Multi-party threshold signature schemes can be part of non-custodial smart wallets, where private keys are shared between parties, and to sign UserOp, a fixed number of key holders must 'meet'. It divides the responsibility for owning the key. Although Shamir's secret sharing algorithm [10] could be performed on mnemonic code to split it between trusted parties, it would be unbelievably difficult for most Ethereum users to understand and perform. One of the goals of account abstraction is to make accounts easier to use. More flexible and robust recovery mechanisms are possible to implement by utilising account abstraction. For example, a wallet can provide a social recovery, where the user can choose trusted guardians (other wallets, for example) to generate a new private key and revoke the old one in case it is lost or stolen. It can also have a delay mechanism enabled when the original wallet owner sees in advance that his account is being recovered (when it does not need to be recovered), so it can prevent malicious behaviour from trusted parties. Furthermore, wallets can utilise standards and recommendations such as NIST SP 800-57 [11], which can greatly increase the level of security. TSS, multisigs and other schemes could exist before ERC 4337 on wallets with contract accounts (for example, Argent wallet), but usually they were custodial or required so-called relayers. Relayers act similarly to paymasters and bundlers, but ERC-4337 can be considered more decentralised and standardized over existing relay solutions.

Trusted sessions with the use of session keys can be a thing in account abstraction. DApps could allow for preapproved transactions with a set of rational restrictions and spending limits. It can be helpful when one needs to perform many small transactions [12]. Moreover, users do not have to worry about the deployment of the account because it will be deployed during the first UserOp [5]. Multicalls can also be performed using a few UserOps created by a single wallet and batched together in one transaction, which can be useful, for example, in swaps.

Bundlers check to see if they can be sure to be compensated. So, participation as a bundler can be risk-free, as user fee payment validation is easy/cheap to perform. ERC-4337 has been designed to greatly mitigate DoS attacks. Bundlers must comply with a set of rules such as those described in ERC-7652 [8]. It assures that performing a DoS attack would be a very expensive task. Separating the validation and execution phases increases protection against DoS attacks, as UserOps of accounts that cannot pay are strictly blocked before execution

begins. ERC-4337 unities are either staked so that malicious behavior costs a lot, or those that are not staked operate under restrictions. To protect the network, entities are subject to a reputation system [5].

Potential for Gas Fee Optimization and Use of ERC-20 Tokens for Payments

Account abstraction allows the use of aggregators that can utilise the Boneh-Lynn-Shacham signature scheme [13], which has also been used in the Ethereum beacon chain protocol [14]. Aggregation can reduce overall gas costs as only one validation is required for all UserOps signatures in an aggregated group. UserOp batching can make gas fees cheaper for senders as there can be deployed multiple calls to contract in a single transaction instead of several. However, some benchmarks indicate that performing a single UserOp costs more than a single EOA transaction [15] [16], which is expected considering the complexity of the UserOperation mechanism. Transaction batching and roll-up could be used for gas optimisation with proposals such as RIP-7560 [17].

The existence of paymasters allows users to pay gas fees in ERC-20 tokens. Potential users don't even have to own ETH to do a transaction; they just need to pick a suitable paymaster or wallet. In some business models, transactions can be paid in full by the paymaster, for example, for new blockchain users who do not understand the whole concept of gas fees, which can scare them off. DApps could cover a fixed amount of gas just to get new customers. Some models could require a subscription payment, which is possible in account abstraction. Those recurring payments on behalf of the user can be automated by ERC-4337 wallet, paymaster, and shop/service integration. Stablecoins could become a payment possibility for vendors that do not want to receive assets whose market value fluctuates quite widely.

Technical and Adoption Challenges

This standard can be viewed as a complex one as it relies on several entities, which, of course makes it difficult to prove its safety. The implementation of account abstraction on eth-infinitism has been the subject to audits done by Openzeppelin [18], and the most recent audit paper (Feb. 2024) shows that all found vulnerabilities have been resolved. However, as the implementation and standards are being further in development, new issues can emerge. One big challenge of this standard is to migrate existing EOA solutions into ERC-4337 smart contract accounts and, within them - all funds that are associated with EOAs. Also, making most of the in-use smart contracts deployed on the chain compatible with ERC-4337 can be challenging, as they have to be upgraded or, if that is not possible, redeployed with code changes. Because gas fees are relatively high on Layer 1 Ethereum, account abstraction is mostly developed on Layer 2. Native account abstraction is implemented on L2 chains, for example, ZkSync [19]. RIP-7560 is a proposal that offers native account abstraction with consensus layer change and is backwards compatible with ERC-4337 [17].

Future developments and potential upgrades to the standard

Making ERC-4337 accounts 'first-class citizens' is certainly the next step for Ethereum developers. ERC-4337 is not native, and Ethereum is currently an EOA-dependent chain. Integration with other proposals, such as EIP-5003 [20], could help with migration to contract accounts. Ensuring that all DApps are compatible with ERC-4337 or a related upgraded standard so that every user can have a smart contract account in the future is probably the ultimate goal [21]. A potential improvement could be to move account abstraction from the high-layer infrastructure to the consensus layer.

Conclusions And Future Work

The functioning of EOA's and that senders must use private keys of ECDSA is embedded in Ethereum protocol. Account abstraction bypasses that and grants users more freedom without losing self-custody. It surely is a great course to achieve fewer wallet dependencies on ECDSA and, in the future, a possible quantum computer-resistant blockchain. User accounts can be managed similarly to how it works in Web2 bank accounts, which can encourage new people to board. The main concern of this standard is that smart accounts are still second-class citizens on Ethereum and that they are not a default account option. There is a chance that in the future, an upgrade to the ERC-4337 standard will be developed, which will change the consensus layer. Therefore, smart contract wallets with signature, nonce and fee abstraction will be a major part of the layer one blockchain.

In general, ERC-4337 can make blockchain interactions user-friendly without losing any of the main Ethereum blockchain principles. It laid the foundation of what Ethereum can become in the future, with Web2 functionalities

on the decentralised, censorship-resistant network. There are many L1 and L2 developers and big services that experiment with account abstraction, including Visa [22].

References

- Cryptocurrency market capitalizations | CoinMarketCap. (2024). CoinMarketCap. https://coinmarketcap.com/
- *Transactions*. (n.d.). Ethereum.org. https://ethereum.org/en/developers/docs/transactions/
- Antonopoulos, A. M., & Wood, G. (2018). *Mastering ethereum: building smart contracts and dapps*. O'reilly Media.
- *Ethereum accounts*. (n.d.). Ethereum.org. https://ethereum.org/en/developers/docs/accounts/
- Buterin, V., Weiss, Y., Tirosh, D., Nacson, S., Forshtat, A., Gazso, K., & Hess, T. (2021). *ERC-4337: account abstraction using alt mempool* (No. 4337). Ethereum Improvement Proposals.
- Buterin, V., Conner, E., Dudley, R., Slipper, M., Norden, I., Bakhta, A. (2019). EIP-1559: *Fee market change for ETH 1.0 chain* (No. 1559). Ethereum Improvement Proposals.
- Buterin. V., (2021, September 29). *ERC 4337: account abstraction without Ethereum protocol changes*. Medium; Infinitism. https://medium.com/infinitism/erc-4337-account-abstraction-without-ethereum-protocol-changes-d75c9d94dc4a
- Weiss, Y., Tirosh, D., Forshtat, A., Nacson, S. (2023). *ERC-7562: Account Abstraction Validation Scope Rules [DRAFT]* (No. 7562). Ethereum Improvement Proposals.
- *Proof-of-stake (PoS)*. Ethereum.org. https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/
- NethermindEth. (2022, September 26). *GitHub - NethermindEth/research-mnemonic: Shamir secret sharing for mnemonic lists*. GitHub. https://github.com/NethermindEth/research-mnemonic
- Barker, E. (2020) NIST special publication 800-57 part 1, revision 5. *Recommendation for Key Management*.
- Ethereum Foundation. (2022, October 12). ELI5: Account Abstraction by Liraz | Devcon Bogotá [Video]. YouTube. https://youtu.be/QuYZWJj65AY?si=6F7klN1J7L0kh-F3
- eth-infinitism. (2022, August 21). *AA-22: Support aggregated signatures and BLS ref. implementation by drortirosh · Pull Request #92 · eth-infinitism/account-abstraction*. GitHub. https://github.com/eth-infinitism/account-abstraction/pull/92
- Edgington, B. (2023). A technical handbook on Ethereum's move to proof of stake and beyond.
- Lin, Z., Wang, T., Zhao, C., Zhang, S., Yang, Q., & Shi, L. (2024, February). A Measurement Investigation of ERC-4337 Smart Contracts on Ethereum Blockchain. In *2024 International Conference on Computing, Networking and Communications (ICNC)* (pp. 1164-1170). IEEE.
- eth-infinitism. (2022). *account-abstraction/reports/gas-checker.txt at develop · eth-infinitism/account-abstraction*. GitHub. https://github.com/eth-infinitism/account-abstraction/blob/develop/reports/gas-checker.txt
- Buterin, V., Weiss, Y., Forshtat, A., Tirosh, D., Nacson, S. (2023). *RIP-7560: Native Account Abstraction* (No. 7560). Rollup Improvement Proposals.
- eth-infinitism. (2022). *account-abstraction/audits at develop · eth-infinitism/account-abstraction*. GitHub. https://github.com/eth-infinitism/account-abstraction/tree/develop/audits
- *Native AA vs EIP 4337 - ZKsync Docs*. (2024). ZKsync Docs. https://docs.zksync.io/build/developer-reference/ethereum-differences/native-vs-eip4337
- Finlay, D., Wilson, S. (2022). *EIP-5003: Insert Code into EOAs with AUTHUSURP [DRAFT]* (No. 5003). Ethereum Improvement Proposals.
- Ethereum Foundation. (2022, October 15). Account Abstraction Panel | Devcon Bogotá [Video]. YouTube. https://youtu.be/WsZBymiyT-8?si=D-SezmujClC8WOLX
- *What is Account Abstraction?* (2021). Visa.com. https://usa.visa.com/solutions/crypto/rethink-digital-transactions-with-account-abstraction.html