Account Abstraction: The Key to Blockchain Reporting

Silviu OJOG, Alina-Andrea MIRON
Bucharest University of Economic Studies, Romania
silviu.ojog@csie.ase.ro, alina.miron@mk.ase.ro

One of the most significant barriers to blockchain adoption is the need for a better user experience. Tasks such as account setup, key management, and transaction handling need to be streamlined to propel the next wave of technological adoption across various sectors. A promising solution to these issues lies in account abstraction, which aims to simplify the user experience by masking the underlying complexities of blockchain technology. The most notable effort to implement account abstraction is Ethereum's ERC-4337, a proposal that addresses these pain points by enhancing flexibility and ease of use for developers and end users alike. This paper explores the account abstraction architecture, its functions and how it can redefine blockchain utility, from smart contracts to smart reporting.

Keywords: *Blockchain, Smart Contract, Security, Ethereum, Exploit, Immutability, Solidity*
DOI: 10.24818/issn14531305/28.3.2024.06

1 Introduction

The emergence of blockchain dates back to 2008, a pivotal period marked by the global financial crisis and the inception of Bitcoin [1]. At its core, blockchain embodies principles that reduce reliance on centralized entities. However, Bitcoin pioneered the blockchain technology primarily for value exchange. Often referred to as the "digital gold" or "gold 2.0", Bitcoin's primary function is a store of value and stems from its strictly limited supply, distinguishing it from other blockchain networks. Despite its technical capability to support smart contracts [2], Bitcoin's adoption in this area remains limited due to its higher cost and slower processing speed. On the other hand, Ethereum [3], the second-largest blockchain network by market cap, capitalized on the smart contract concept and became a store of value.

The "blockchain trilemma", a term coined by Vitalik Buterin, one of the founders of Ethereum, refers to the inherent challenge blockchain networks face in balancing three fundamental attributes: security, decentralization, and scalability. These attributes represent core principles often in tension with each other, making it challenging to achieve optimal performance in all three areas simultaneously.

Scalability is the ability of a blockchain network to handle an increasing number of transactions or users without compromising performance. This includes transaction throughput, confirmation times, and overall network efficiency. Scalability comes with costs, some transactional, such as gas fees, and others less visible technical knowledge. Many times, that cost is difficult for a user to bear. That is why the adoption of blockchain by users beyond early adopters is challenging.

2 Security and UX considerations

Digital signatures provide protections within the blockchain context, notably integrity assurances. They guarantee that a transaction remains unaltered from its transfer from the initiator to its long-term storage on each blockchain node. The system also benefits from authentication and non-repudiation, thereby establishing the origin of a transaction as that of the account owner or an authorized user [4].

The effectiveness of our digital signature algorithm is solely dependent on the confidentiality of this private key. Possessing this key allows the generation of valid digital signatures and the creation of legitimate blockchain transactions.

Therefore, the security of a blockchain account is significantly dependent on safeguarding the private key associated with that account. Whoever possesses the private key can access any functionality within the blockchain, most significantly transferring assets. A private key is considered compromised if revealed even for a split second. Therefore, private key management in blockchain represents a paradigm shift from the real world, where possessing a key does not serve as definitive proof of ownership. Just like possessing the keys to a house grants access, it does not automatically prove ownership. In the real world, other participants may possess keys that allow access without asserting ownership over the underlying assets or resources.

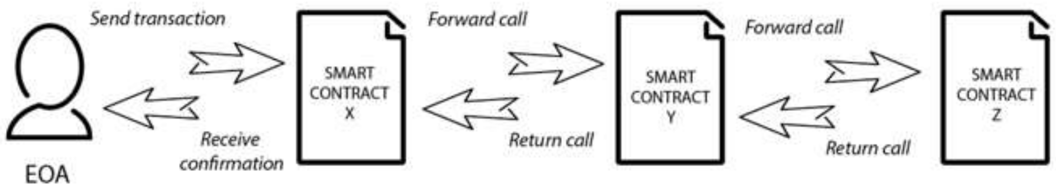

Fig. 1. Smart Contract Communication

The challenge lies in maintaining a delicate balance for the average user. Strong security measures are essential to prevent unauthorized access to sensitive information and funds. Such measures may include robust authentication methods such as passwords, biometrics, and two-factor authentication (2FA). While it is critical to protect the private key, it is equally essential to ensure its usability. Ideally, the most secure method of safeguarding a private key might involve writing it on paper. However, such measures render the private key inaccessible for transactional use. Therefore, the aim is to enable intermediate users to access their private keys while preventing unauthorized access by attackers. On the other hand, users expect fast and seamless transactions without unnecessary delays or complications. Optimizing transaction processing times and providing clear feedback on transaction status can enhance user satisfaction.

3 The current Ethereum Architecture

As a smart contract blockchain platform, Ethereum supports two types of accounts: Externally Owned Accounts (EAO) and Contract Accounts (CA) [5]. There are crucial differences regarding the ability to interact with these two types of accounts and what information they store.

A Contract Account is, in fact, a smart contract that contains the instructions in the form of an opcode that the Ethereum Virtual Machine EVM runs, while the EAO does not. Once a CA is deployed on a blockchain, anyone can view and interact with its code. Certain features may be restricted for interaction, but they are visible.

A smart contract receives instructions through transactions. They can provide the input to smart contracts, which are run on the EVM and can be initiated only by an EAO, but can be forwarded by a CA [6].

A transaction occurs when a new contract is deployed on the blockchain or when tokens are transferred between two externally owned accounts or from one externally owned account to a contract account. An external actor controls an EAO through private keys while a CA has no private key. Smart contracts, and therefore the CAs can have an owner entity with elevated privileges to access and edit data from a contract.

Nevertheless, transactions do not end in a database immediately. They are first submitted to blockchain participants called nodes, and new transactions are constantly emitted between nodes. The sum of all transactions not inserted in the blockchain yet form the memory pool, or mempool in short.

Every transaction is accompanied by a gas limit, which estimates the amount (gas) the transaction will spend. The gas limit imposed by the transaction creator, and it depends on the complexity of the operation.

4 Account Abstraction implementation

The Ethereum ecosystem has two mechanisms for proposing and standardizing changes and enhancements of the Ethereum protocols: EIPs (Ethereum Improvement Proposals) and ERCs (Ethereum Request for Comments). EIPs are detailed design documents outlining new features or modifications to Ethereum's core protocol, network, interfaces, or application standards. Each EIP undergoes rigorous peer review within the Ethereum community before potential implementation [7]. On the other hand, ERCs focus specifically on proposing standards for Ethereum functionalities such as token protocols (e.g., ERC-20 for fungible tokens or ERC-721 for non-fungible tokens). ERCs provide formal specifications and guidelines, fostering interoperability and compatibility among Ethereum-based applications. Both EIPs and ERCs have significant functions in shaping Ethereum's technical evolution and standardization, contributing to its continuous development and adoption within the blockchain ecosystem.

The main difference between ERC and EIP is the network's implementation method. For the first, it is not necessary to change the protocol, i.e., create a fork. A fork is a concept from the DevOps world, from the git tool. It means creating two different code bases. There have been several such splits in history, the most recent being the transition to the new type of consensus, the proof of stake consensus, in which the Ethereum PoW (Proof of Work) blockchain was created, which has the same blockchain as Ethereum.

In the blockchain context, "account abstraction" refers to simplifying or concealing technical processes from the end user. ERC 4337 proposes a new architecture for abstracting out the mechanism of creating an Externally Owned Account (EOA), signing and sending transactions to the blockchain. This approach aims to shield users from the intricacies of underlying technologies, enabling them to benefit without needing a deep understanding of the operational details. In the future, ERC 4337 may turn into EIP.

Account abstraction represents a transformative concept within blockchain technology. It reshapes the traditional notion of accounts by treating each one as a smart contract rather than a conventional Externally Owned Account (EOA). This shift introduces possibilities where smart contracts can embody custom logic tailored to specific needs and use cases. ERC 4337 includes several components, detailed in the following sections [8].

4.1 User Operations

On User Operation represents the action the user wants his wallet to perform ERC 4337 introduces the UserOperations mempool, an alternative memory pool of transaction, alongside traditional transaction handling. This new mempool specifically caters to UserOperations, which are pseudo transaction objects generated when users engage with decentralized applications (dApps). Instead of routing through the conventional transaction mempool, UserOperations are directed to the UserOperations mempool. UserOperations allow grouping multiple actions into a single operation. By signing this combined operation, users delegate its execution to the Ethereum network. UserOperations do have a structure similar to Ethereum transactions while integrating specific logic defined by ERC-4337. Similar to standard transactions, UserOperations contain familiar fields like sender, recipient, calldata, maxFeePerGas, maxPriorityFee, signature, and nonce. However, they also feature additional fields, elaborated upon in subsequent sections, as multiple validations are needed before transaction execution [8].

4.2 Bundlers

Following UserOperation preparation, Bundlers are entities that integrate these operations into the Ethereum network. They gather multiple UserOperations as intermediaries before submitting them to the network. Bundlers can function as validators or MEV (Miner Extractable Value) searchers, ensuring efficient transation processing within the network. Bundlers actively monitor the UserOperations mempool for incoming

UserOperations, consolidating them into Bundle Transactions by aggregating multiple UserOperations. Subsequently, every Bundle transaction invokes the handleOps function within the EntryPoint contract.

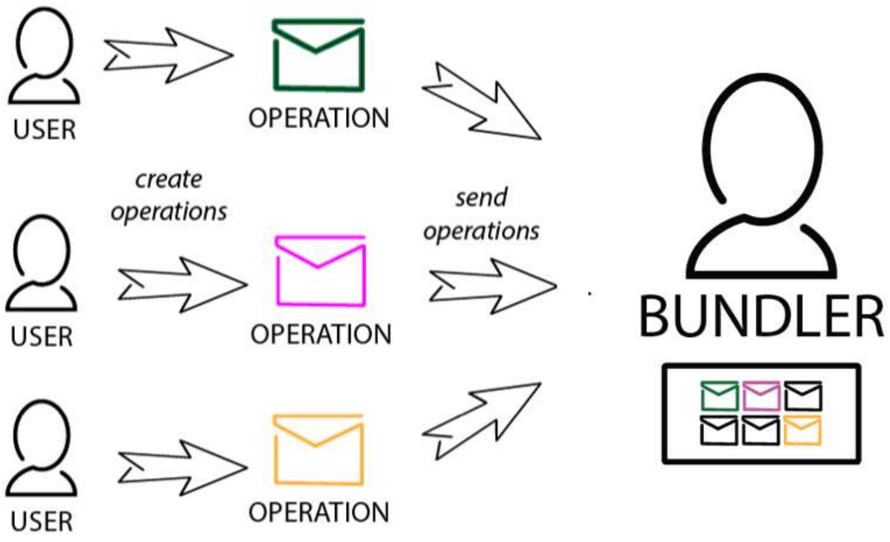

Fig. 2. Entry Point Contract

4.3 Entry Point

Operating as the gatekeeper for the Ethereum network, the EntryPoint smart contract unpacks and executes UserOperations submitted by Bundlers. The EntryPoint contract acts as a singleton on the whole Ethereum Network. If an operation encounters an issue, the EntryPoint can reverse its actions, safeguarding transaction integrity and reliability. The EntryPoint contract operates as a singleton, meaning it exists as a singular instance. Its primary role revolves around validating and executing Bundle Transactions. Prior to integrating a UserOperation into a Bundle Transaction, Bundlers execute a simulateValidation function call within the EntryPoint contract. The UserOperation is omitted from the Bundle Transaction if the validation proves unsuccessful [8].

4.4 Factory Contract

A Factory contract calls to create a wallet contract for the user. It has a double role. Firstly: it solves the cold start problem when a wallet contract is not created and must be deployed. Secondly, a new smart contract wallet could be deployed in case of a zero-day vulnerability within the smart contract wallet. The Ethereum Storage Contract Inheritance must be followed.

Factory Contracts must be deployed prior to the Smart Contract and have sufficient funds to deploy a new contract.

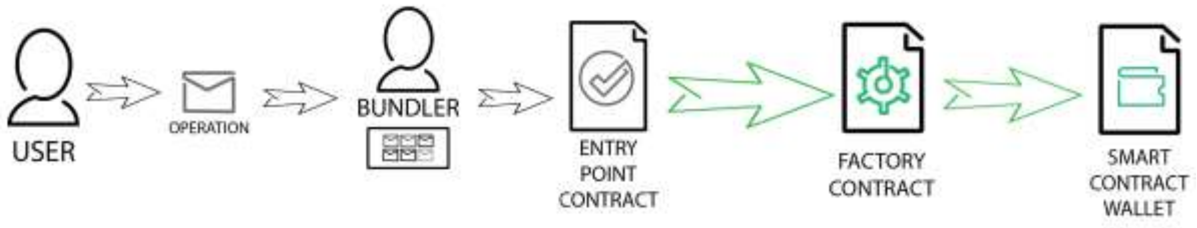

Fig. 3. Factory Contract

4.5 Smart Contract Account (Wallet Account)

Conceptualized as an automated assistant within the Ethereum network, contract accounts differ from standard accounts in that they autonomously execute actions based on received instructions, such as those from a user operation. They facilitate interactions with other contracts, asset management, and decision-making based on programmed logic, streamlining complex Ethereum transactions through automation.

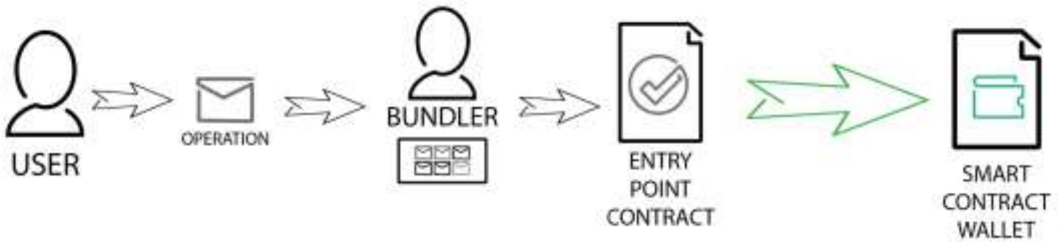

Fig. 4. Smart Contract Wallet

The responsibilities of the account contract include: validating that the execution call originates from a legitimate EntryPoint contract, verifying the validity of the signature using a preferred validation mechanism and addressing missingAccountFunds, which represents the additional funds needed to execute a UserOperation if the account's deposit in the EntryPoint contract is insufficient.

The validateUserOp function primarily focuses on signature validation and ensuring adequate funds to cover gas costs without executing the actual call data operations. To maintain validation integrity between the validation and execution phases, certain restrictions are placed on the function: Prohibition of specific opcodes like BLOCKHASH and TIMESTAMP to prevent value changes between phases. Restriction of storage access to only the account's associated storage and relevant contract storage, ensuring data consistency.

Additionally, the EntryPoint contract requests the maximum gas amount, retaining the surplus for future operations. The account is only required to cover missingAccountFunds, which is not provided by the EntryPoint contract for this specific scenario.

4.6 Paymaster

The paymaster is an optional component capable of covering transaction fees on behalf of transactions. It commits to reimbursing the Bundler for gas costs under specified conditions outlined in the associated smart contract. It must also maintain a locked stake to prevent malicious actors' potential abuse of the system. This staking requirement is a deterrent against paymasters who might initially agree to cover transaction costs but later reject them, potentially leading to a denial-of-service (DoS) attack.

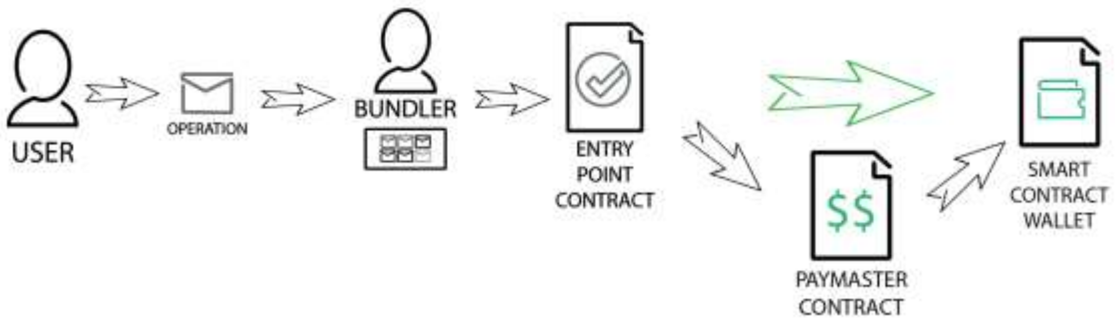

Fig. 5. Paymaster

A reputation system similar to traditional web environments is implemented to ensure accountability. Paymasters with significant transaction failures face temporary bans, deterring malicious behavior. Each Bundler maintains individual reputation tracking systems, allowing for customized approaches to managing paymaster reputations.
It is important to note that even if a paymaster behaves maliciously, their stake is not forfeited, distinguishing this system from others that penalize malicious actors by seizing their stakes. However, there are exceptions to the staking rule: Paymasters may not need to stake if they pass validation but fail execution due to storage changes occurring between the two steps, typically caused by multiple operations altering the same storage. Paymasters who do not utilize storage or solely rely on the account's storage and not their own may also be exempt from staking requirements. Successful validation implies a low likelihood of execution failure, minimizing the risk of malicious behavior from the Paymaster.

5 Implementing Account Abstraction

The core idea of account abstraction extends beyond simplicity, offering nuanced capabilities that will unfold progressively. A vital feature of this paradigm is the separation of the signer from the account itself, liberating transactions from the constraints of a single entity responsible for signing messages.
Decoupling the signer from the account opens doors to various innovations, including enabling multiple signers for a single account, searching for alternative signature schemes beyond the Elliptic Curve Digital Signature Algorithm (ECDSA), implementing distinct validation methods for different transaction types or potentially integrating biometric authorization methods used outside of blockchains like FaceID or TouchID. Figure 7 bellow depicts such an authorization flow.

.

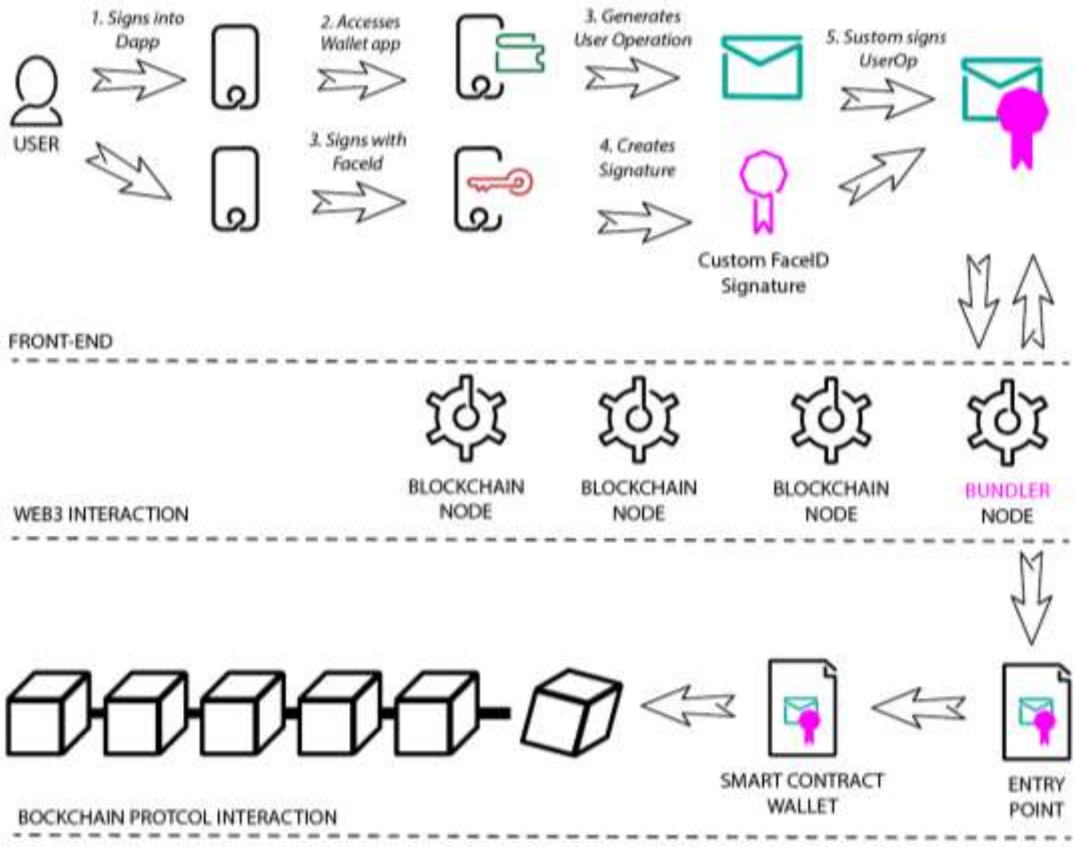

Fig. 6. Custom signature for Smart Contracts

These capabilities empower smart contract accounts to define their transaction validation criteria, fee payment mechanisms, and transaction initiation processes based on predefined logic. The flexibility offered by account abstraction invites creativity and innovation in blockchain development. Several notable use cases facilitated by Account Abstraction include innovative approaches to gas fee management, transaction batching, access control, gas sponsorships, and custom signature schemes.
ERC 4337 proposes a dynamic architecture in which specific actors such as Paymasters are not compulsory to exist. Factory Contracts, for example, are very rare in usage flow, often only once when deploying the contract. Figure 8 shows, using the flow chart, the different flows that a user transaction can take until it is rejected or executed successfully and has repercussions in the blockchain.

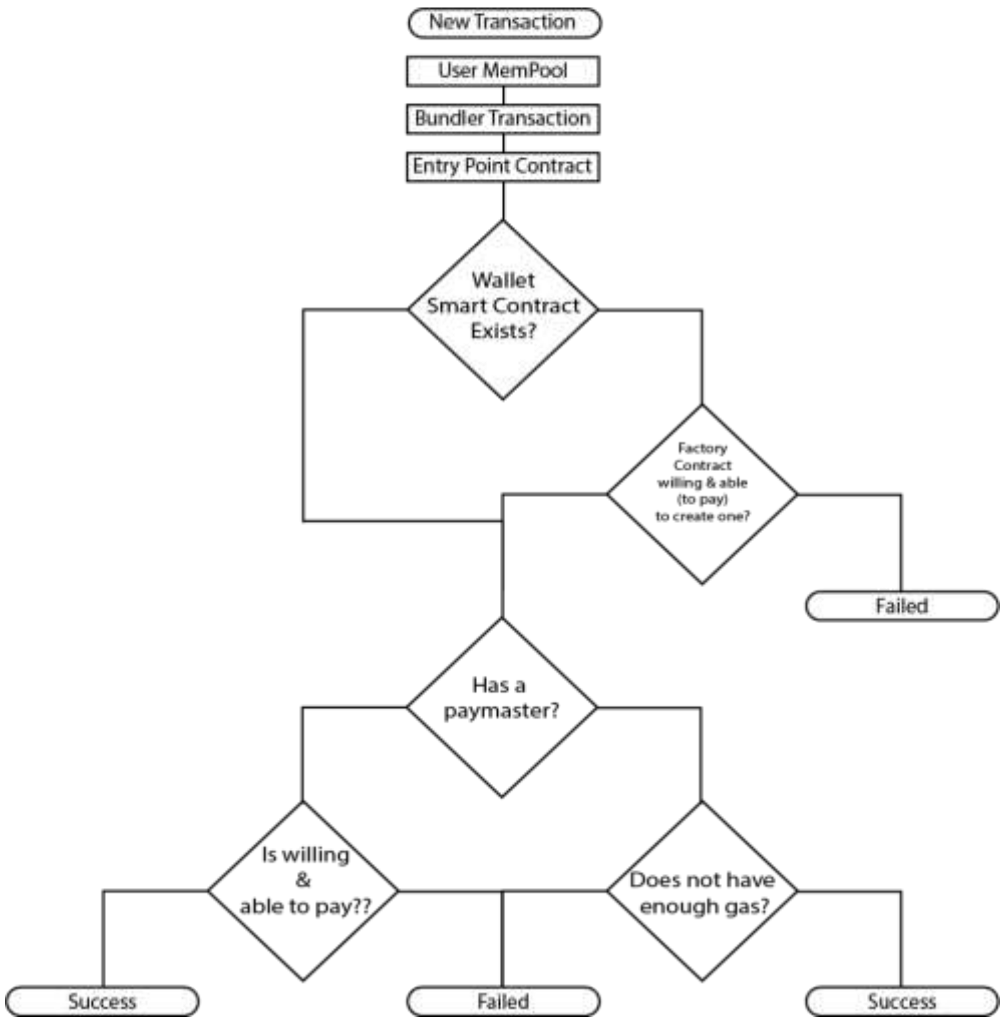
New Transaction
User MemPool
Bundler Transaction
Entry Point Contract
Wallet Smart Contract Exists?
Factory Contract willing & able (to pay) to create one?
Failed
Has a paymaster?
Is willing & able to pay??
Does not have enough gas?
Success
Failed
Success

Fig. 7. Assessing Smart Contracts Components

If we think about what blockchain means, at the root, we see that we are dealing with a distributed ledger technology. Historically, ledgers have been used to account for and report various assets. Blockchain followed a similar path by accounting financial transactions with the spring of Bitcoin and gradually incorporating more sophisticated assets. Some of those assets may create entirely new markets and economies, such as the Decentralized Finance (DEFI) market [11].

The first step towards trust in technology has been achieved. Abstracting the account, comes to ease the process. In the world, reporting is an arduous process. A report is nothing but a transaction according to the rules. For example, in a reporting process we have the one who reports and the one to whom it is reported. To make reporting easier, the friction must be lower. That is, the person to whom it is reported must be the one who sets the mechanism in motion, that is, to control the creation of the Smart Contract Account by managing the Factory Account. This empowers the reporting process, giving control to the reporting authority. As stated earlier, interacting with a contract requires transaction costs [12]. The respective costs can be borne by the authority requesting the respective report.

7 Conclusions and Future Work

Blockchain is an ever-evolving technology, with challenges and risks constantly rising. It has the potential to disrupt many online industries and business models. Smart contract security is a particularly important piece of the blockchain puzzle as it handles the creation, storage and distribution of valuable assets. Smart contracts are immutable, public, and fully open for interaction; thus, new approaches are needed. Researchers and developers must consider it and the adversary environment and prepare accordingly by adopting the best standards and performing smart contract security audits [13].

In order to accelerate technology adoption, the blockchain research community must continue to provide explicit models for creating, securing, error handling, and interacting with smart contracts. Mass adoption is likely reached when many low-level interactions with smart contracts and blockchains are abstracted. Several example abstraction implementations that are compliant with the ERC-4337 standard are needed.

New users will likely emerge as the industry progresses in creating new examples. Attracting new users will lead to the discovery of new use cases, which will mature the blockchain industry and create much-anticipated market disruptions.

References

[1] Bitcoin: A Peer-to-Peer Electronic Cash System (2008) - Satoshi Nakamoto. Available: https://bitcoin.org/bitcoin.pdf
[2] N. Szabo. Smart Contracts: Building Blocks for Digital Markets. Available: http://www.truevaluemetrics.org/DBpdfs/BlockChain/Nick-Szabo-Smart-Contracts-Building-Blocks-for-Digital-Markets-1996-14591.pdf (1996).
[3] Ethereum Whitepaper: A Next-Generation Smart Contract and Decentralized Application Platform. Available: https://ethereum.org/en/whitepaper/ (2014).
[4] V. Buterin. Ethereum: A Secure Decentralised Generalised Transaction Ledger. Shanghai Version 57d59cd. Available: https://ethereum.github.io/yellowpaper/paper.pdf (2024/07/23).
[5] X. Wu, Z. Zou, D. Song, Learn Ethereum. 2nd edn. Publisher, Location.
[6] S. Ojog, An Overview of Security Issues in Smart Contracts on the Blockchain. In: Proceedings of 21st International Conference on Informatics in Economy (IE 2022), vol. SIST, no. 321, pp. 51–63. Springer, Heidelberg (2023).
[7] Ethereum Improvement Proposals. Available: https://eips.ethereum.org/EIPS/eip-4337, last accessed 2023/05/14.
[8] Ethereum Foundation. Available: https://www.erc4337.io/, last accessed 2023/05/14.
[9] Open Zeppelin: ERC-4337 Account Abstraction - Incremental Audit. Available: https://blog.openzeppelin.com/erc-4337-account-abstraction-incremental-audit.
[10] D.J. Bernstein, T. Lange, P. Schwabe. High-speed high-security signatures. Available: https://ed25519.cr.yp.to/ed25519-20110926.pdf, last accessed 2023/05/14.
[11] S. Ojog. The Emerging World of Decentralized Finance. Informatica Economică vol. 25, no. 4, pp. 2021 (2022).

[12] M. Araoz, D. Brener, F. Giordano, S. Palladino, T. Paivinen, A. Gozzi, F. Zeoli. Zeppelin OS: An Open-Source, Decentralized Platform of Tools and Services on Top of the EVM to Develop and Manage Smart Contract Applications Securely. Available: https://openzeppelin.com/assets/zeppelin_os_whitepaper.pdf (2017).

[13] Solidity Security Patterns. Available: https://github.com/fravoll/solidity-patterns, last accessed 2023/05/14.

Silviu OJOG has graduated the "Gh. Asachi" Technical University, in 2013, in Iasi Romania BSc in Applied Electronics. He graduated University of Bucharest, Romania MSc in Software Engineering, 2016. He is currently enrolled as a PhD Student Economic Informatics, Bucharest University of Economic Studies. He holds a certification in new venture leadership from the Massachusetts Institute of Technology, USA, following a study program in Brisbane, Australia.

Alina-Andrea MIRON graduated in Public Administration from the National School of Political and Administrative Studies in 2012 in Bucharest, Romania. She holds two master's degrees, one in Public Sector Management at the National School of Political and Administrative Studies in 2014 and the other in Online Marketing at the Academy of Economic Studies in Bucharest, Romania, in 2021. She is currently enrolled as a PhD student in Marketing, and her thesis is related to social responsibility communication. She is also an associate professor teaching PR.