title: Account abstraction description: An overview of Ethereum's plans to make user accounts simpler and safer lang: en summaryPoints: - Account abstraction makes it much easier to build smart contract wallets - Smart contract wallets make it much easier to manage access to Ethereum accounts - Lost and exposed keys can be recovered using multiple backups

Account abstraction {#account-abstraction}

Users interact with Ethereum using **externally owned accounts (EOAs)**. This is the only way to start a transaction or execute a smart contract. This limits how users can interact with Ethereum. For example, it makes it difficult to do batches of transactions and requires users to always keep an ETH balance to cover gas.

Account abstraction is a way to solve these problems by allowing users to flexibly program more security and better user experiences into their accounts. This can happen by upgrading EOAs so they can be controlled by smart contracts, or by upgrading smart contracts so they can initiate transactions. These options both require changes to the Ethereum protocol. There is also a third path involving adding a second, separate transaction system to run in parallel to the existing protocol. Regardless of the route, the outcome is access to Ethereum via smart contract wallets, either natively supported as part of the existing protocol or via an add-on transaction network.

Smart contract wallets unlock many benefits for the user, including:

- define your own flexible security rules
- recover your account if you lose the keys
- share your account security across trusted devices or individuals
- pay someone else's gas, or have someone else pay yours
- batch transactions together (e.g. approve and execute a swap in one go)
- more opportunities for dapps and wallet developers to innovate on user experiences

These benefits are not natively supported today because only externally-owned accounts (EOAs) can start transactions. EOAs are simply public-private key pairs. They work like this:

- if you have the private key you can do *anything* within the rules of the Ethereum Virtual Machine (EVM)
- if you do not have the private key you can do *nothing*.

If you lose your keys they can't be recovered, and stolen keys give thieves instant access to all the funds in an account.

Smart contract wallets are the solution to these problems, but today they are difficult to program because in the end any logic they implement has to be translated into a set of EOA transactions before they can be processed by Ethereum. Account abstraction enables smart contracts to initiate transactions themselves, so that any logic that the user wishes to implement can be coded into the smart contract wallet itself and executed on Ethereum.

Ultimately, account abstraction improves support for smart contract wallets, making them easier to build and safer to use. In the end, with account abstraction, users can enjoy all the benefits of Ethereum without knowing or caring about the underlying technology.

Beyond seed phrases {#beyond-seed-phrases}

Today's accounts are secured using private keys that are calculated from seed phrases. Any person who has access to a seed phrase can easily discover the private key protecting an account and gain access to all the assets it protects. If a private key and seed phrase are lost, they can never be recovered and the assets they control are frozen forever. Securing these seed phrases is awkward, even for expert users and seed phrase phishing is one of the most common ways users get scammed.

Account abstraction will solve this problem by using a smart contract to hold assets and authorize transactions. These smart contracts can then be decorated with custom logic to make them as secure and tailored to the user as possible. Ultimately, you still use private keys to control access to your account, but with safety nets that make them easier and safer to manage.

For example, backup keys can be added to a wallet so that if you lose or accidentally expose your main key, it can be replaced with a new, secure one with permission from the backup keys. You might secure each of these keys in a different way, or split them across trusted guardians. This makes it much harder for a thief to gain full control over your funds. Similarly, you can add rules to the wallet to reduce the impact if your main key gets compromised, for example you might allow low value transactions to be verified by a single signature, whereas higher value transactions require approval from multiple authenticated signers. There are other ways smart contract wallets can help you to thwart thieves too, for example a whitelist can be used to block every transaction unless it is to a trusted address or verified by several of your pre-approved keys.

Examples of security logic that can be built into a smart contract wallet:

- **Multisig authorization**: You can share authorization credentials across multiple trusted people or devices. Then the contract can be configured so that transactions of more than some preset value require authorization from a certain proportion (e.g. 3/5) of the trusted parties. For example, high value transactions might require approval from both a mobile device and a hardware wallet, or signatures from accounts distributed to trusted family members.
- **Account freezing**: If a device is lost or compromised the account can be locked from another authorized device, protecting the user's assets.
- **Account recovery**: Lost a device or forgotten a password? In the current paradigm this means your assets could be frozen forever. With a smart contract wallet, you can set some pre-approved accounts that can authorize new devices and reset access.
- **Set transaction limits**: Specify daily thresholds for how much value can be transferred from the account in a day/week/month. This means if an attacker does gain access to your account they can't drain everything at once and you have opportunities to freeze and reset access.
- **Create whitelists**: Only allow transactions to certain addresses that you know to be safe. This means that *even if* your private key was stolen, the attacker could not send funds to non-whitelisted destination accounts. These whitelists would require multiple signatures to change them so that an attacker couldn't add their own address to the list unless they had access to several of your backup keys.

Better user experience {#better-user-experience}

Account abstraction allows for a **better overall user experience** as well as **improved security** because it adds support for smart contract wallets at the protocol level. The most important reason for this is that it will provide developers of smart contracts, wallets and applications with much more freedom to innovate on the user experience in ways that we may not yet be able to anticipate. Some obvious improvements that will come along with account abstraction include bundling of transactions for speed and efficiency. For example, a simple swap should be a one-click operation, but today it requires signing multiple transactions for approving spending of individual tokens before the swap is executed. Account abstraction removes that friction by allowing transaction bundling. Furthermore, the bundled transaction could approve precisely the right value of tokens required for each transaction and then revoke the approvals after the transaction completes, providing additional security.

Gas management is also much improved with account abstraction. Not only can applications offer to pay their users' gas fees, but gas fees can be paid in tokens other than ETH, freeing users from having to maintain an ETH balance for funding transactions. This would work by swapping the user's tokens for ETH inside the contract and then using the ETH to pay for gas.

Gas management is one of the primary frictions for users of Ethereum, mainly because ETH is the only asset that can be used to pay for transactions. Imagine you have a wallet with a USDC balance, but no ETH. You can't move or swap those USDC tokens because you can't pay gas. You can't swap the USDC for ETH either, because that in itself costs gas. You would have to send more ETH to your account from an exchange or another address to solve the problem. With smart contract wallets, you can simply pay gas in USDC instead, freeing your account. You don't have to keep an ETH balance in all your accounts anymore.

Account abstraction also allows dapp developers to be creative with gas management. For example, you might be able to start paying your favourite DEX a fixed fee each month for unlimited transactions. Dapps might offer to pay all your gas fees on your behalf as a reward for using their platform, or as an onboarding offer. It will be much easier for developers to innovate on gas when smart contract wallets are supported at the protocol-level.

Trusted sessions are also potentially transformative for user experiences, especially for applications like gaming, where large numbers of small transactions might need approval in a short time. Individually approving each transaction would break the gaming experience, but permanent approval is unsafe. A smart contract wallet could approve certain transactions for a fixed time, up to a specific value or only to certain addresses.

It is also interesting to consider how purchases could change with account abstraction. Today, each transaction has to be approved and executed from a wallet pre-funded with a sufficient amount of the correct token. With account abstraction, the experience could be more like familiar online shopping where a user could fill a "basket" with items and click once to purchase all at once, with all the logic required handled by the contract, not the user.

These are just a few examples of how user experiences could be levelled up by account abstraction, but there will be many more that we haven't imagined yet. Account abstraction frees developers from the constraints of present-day EOAs, allows them to bring the good aspects of web2 into web3 without sacrificing self-custody and to hack creatively on inventive new user experiences.

How will account abstraction be implemented? {#how-will-aa-be-implemented}

Smart contract wallets exist today but are challenging to implement because the EVM does not support them. Instead, they rely on wrapping relatively complex code around standard Ethereum transactions. Ethereum can change this by allowing smart contracts to initiate transactions, handling the necessary logic in Ethereum smart contracts instead of off-chain. Putting logic into smart contracts also increases Ethereum's decentralization since it removes the need for "relayers" run by wallet developers to translate messages signed by the user to regular Ethereum transactions.

EIP-2771 introduces the concept of meta-transactions that allow third parties to pay for a user's gas costs without making changes to the Ethereum protocol. The idea is that transactions signed by a user get sent to a `Forwarder` contract. The forwarder is a trusted entity that verifies that transactions are valid before sending them on to a gas relay. This is done off-chain, avoiding the need to pay gas. The gas relay passes the transaction on to a `Recipient` contract, paying the necessary gas to make the transaction executable on Ethereum. The transaction is executed if the `Forwarder` is known and trusted by the `Recipient`. This model makes it easy for developers to implement gasless transactions for users.

EIP-4337 is the first step towards native smart contract wallet support in a decentralized way*without requiring changes to the Ethereum protocol*. Instead of modifying the consensus layer to support smart contract wallets, a new system is added separately to the normal transaction gossip protocol. This higher level system is built around a new object called a `UserOperation` that package up actions from a user along with the relevant signatures. These`UserOperation` objects then get broadcast into a dedicated mempool where validators can collect them into a "bundle transaction". The bundle transaction represents a sequence of many individual `UserOperations` and can be included in Ethereum blocks just like a normal transaction, and would be picked up by validators using a similar fee-maximizing selection model.

The way wallets work would also change under EIP-4337. Instead of each wallet re-implementing common but complex safety logic, those functions would be outsourced to a global wallet contract known as the "entry point". This would handle operations such as paying fees and executing EVM code so that wallet developers can focus on providing excellent user experiences.

Note the EIP 4337 entry point contract was deployed to Ethereum Mainnet on 1st March 2023. You can see the contract on Etherscan.

EIP-2938 aims to update the Ethereum protocol by introducing a new transaction type,`AA_TX_TYPE` that includes three fields: `nonce`, `target` and `data`, where `nonce` is a transaction counter, `target` is the entry point contract address and `data` is EVM bytecode. To execute these transactions, two new instructions (known as opcodes) have to be added to the EVM: `NONCE` and `PAYGAS`. The `NONCE` opcode tracks the transaction sequence and`PAYGAS` calculates and withdraws the gas required to execute the transaction from the contract's balance. These new features allow Ethereum to support smart contract wallets natively as the necessary infrastructure is built in to Ethereum's protocol.

Note that EIP-2938 is currently not active. The community is currently favoring EIP-4337 because it does not require changes to the protocol.

EIP-3074 aims to update Ethereum's externally-owned accounts by allowing them to delegate control to a smart contract. This means smart contract logic could approve transactions originating from an EOA. This would allow features such as gas-sponsoring and batched transactions. For this to work, two new opcodes have to be added to the EVM: `AUTH` and `AUTHCALL`. With EIP-3074 the benefits of a smart contract wallet are made available *without needing a contract* - instead, a specific type of stateless, trustless, non-upgradeable contract known as an "invoker" handles the transactions.

Note that EIP-3074 is currently not active. The community is currently favoring EIP-4337 because it does not require changes to the protocol.

Current progress {#current-progress}

Smart contract wallets are already available, but more upgrades are required to make them as decentralized and permissionless as possible. EIP-4337 is a mature proposal that does not require any changes to Ethereum's protocol, so it is possible that this could be implemented quickly. However, upgrades that alter Ethereum's protocol are currently not in active development, so those changes may take much longer to ship. It is also possible that account abstraction is achieved well enough by EIP-4337 that no protocol changes are ever required.

Further reading {#further-reading}

- erc4337.io
- Account abstraction panel discussion from Devcon Bogota
- "Why account abstraction is a game changer for dapps" from Devcon Bogota
- "Account abstraction ELI5" from Devcon Bogota
- Vitalik's "Road to Account Abstraction" notes
- Vitalik's blog post on social recovery wallets
- EIP-2938 notes
- EIP-2938 documentation
- EIP-4337 notes
- EIP-4337 documentation
- EIP-2771 documentation
- "Basics of Account Abstraction" -- What is Account Abstraction Part I