Why Account Abstraction is a Game-Changer for Dapps

Scaling self-custody

Julien Niset
co-founder@Argent
Self-Custody Matters!

Blockchains enable true digital ownership. But you don’t own an asset if you don’t have custody!
But Self-Custody is hard

stazie @stazie · Jul 31
1/9 I lost my punks and a bunch of ETH ↓

232  t 356  1K

stazie @stazie · Jul 31
2/9 I was lying in bed yesterday evening, mind was very foggy, casually browsing. Saw this bot in Discord and clicked the link. The site looked like Cryptopunks, and had a popup that looked like Metamask...

shan @shan_crypt0 · Aug 4
my wallets were drained today gg

here's ur reminder to employ the best opsec

Nexus Mutual @NexusMutual · Dec 14, 2020
At 9:40am this morning @HughKarp's personal address was attacked and drained by a member of the mutual. Only Hugh's address was affected in this targeted attack and there is no subsequent risk to Nexus Mutual or any members.

EtherTransactionHash (Txhash) Details | Eth... Ethereum (ETH) detailed transaction info for txhash ...

shanscan.io

Ben Hunt @EpsilonTheory · Aug 10
If my tradfi account is hacked, my loss is $0.

If my credit card is stolen, my loss is $50.

If my DeFi account is exploited, my loss is everything.

This isn't a DeFi growing pain. This is a permanent, structural feature of decentralization. Can we just be honest about that?
The problem: Ethereum Accounts

Two types of Accounts on Ethereum:

- Externally Owned Account (EOA) → can initiate transactions
- Contract Account (CA) → can contain logic

An (EOA) Account has:

- an **address** for identification
- a **nonce** to make sure transactions are unique
- a **balance** in ETH to pay fees

A user **owns** an EOA through a pair of cryptographic keys (aka a **Signer**):

- The address of the account is derived from the **public key**
- Transactions from the account must be signed by the **private key**

The signature scheme of Ethereum is ECDSA on the elliptic curve secp256k1.
The problem: Ethereum Accounts

Signer

\((k_{\text{priv}}, k_{\text{pub}})\)

EOA

\(\text{address} = \text{keccak}(k_{\text{pub}})[12:32]\)

\(\text{nonce} + \text{balance}\)

EVM logic to \textbf{validate} transactions \(f(k_{\text{priv}})\)

EVM logic to \textbf{execute} transactions

User

Ethereum

ECDSA on Secp256k1
The problem: Ethereum Accounts

Signer
(k_{priv}, k_{pub})

EOA
address = keccak(k_{pub})[12:32]
nonce + balance
EVM logic to validate transactions f(k_{priv})
EVM logic to execute transactions

User Ethereum

The concept of **Signer** and the concept of **Account** are merged.

Authorised to spend your tokens

Holds your tokens
EOAs are the Problem

If the Signer **IS** the Account, and vice versa, then...

- If you lose your Signer you lose your Account!
- If I have your Signer I have your Account!

🙊

The entire security of Ethereum relies on users managing a single secret ($k_{priv}$)

⚠

There is no safety net; users can’t make mistakes

This will not work for mainstream adoption!
Can we do better?
Account Abstraction!

Decouple the relation between Account and Signer

→ Signer ≠ Account

The Account is a smart-contract that defines what a valid transaction is:

- Different Signature scheme?
- Different elliptic curve?
- Multiple Signers?
- Signer can be replaced?

Account Abstraction means every account is a Smart-contract wallet that can initiate transactions and pay the fee.

No more EOAs!!!
Smart-contract wallets: program smart-contracts to secure user assets. Enables social recovery, on-chain fraud monitoring, etc.

EIP-3074: make existing EOAs behave like smart-contracts by allowing users to delegate control of their EOA to a contract.

EIP-86: introduces “forwarding contract” to abstract signature verification and nonce.

EIP-2938: introduces new AA transaction to enable smart-contracts to act as top-level accounts.

EIP-4337: decentralises the infrastructure needed to write and operate smart-contract wallets. No protocol change needed!
EIP 4337

1. User sends **UserOperation** to higher level mempool

2. **Bundler** bundles UserOperations into a bundle transaction.

3. Bundle transaction is sent to the **EntryPoint** contract. Bundler pays the transaction fee.
EIP 4337

1. For each UserOperation the EntryPoint contract calls the `validateUserOp` method of the target wallet. The wallet validates the operation (signature, nonce, etc) and pays the fee.

2. For each (validated) UserOperation the EntryPoint contract calls an execution method that can interpret the calldata and `execute` the call(s).
What’s next for Account Abstraction?

Each of these EIPs (86, 2938, 3074, 4337) delivers some of the features of Account Abstraction.

EIP-4337 comes close, but like smart-contract wallets, 4337 wallets remain second class citizens on a blockchain designed for EOAs.

What we need is to bring EIP-4337 from the application layer to the protocol layer and remove EOAs!

L2 is the perfect opportunity to fix the limitations of Ethereum and bring AA to users.
Native Account Abstraction on StarkNet (and ZkSync)
The IAccount interface*

namespace IAccount {
    struct Call {
        to: felt,
        selector: felt,
        data_len: felt,
        data: felt*
    }
    func __validate__ (calls_len: felt, calls: Call*) -> (success: felt){}  
    func __validate_declare__ (class_hash: felt){}  
    func __validate_deploy__ (class_hash: felt, ctr_arg_len: felt, ctr_args: felt*, salt: felt)  
    func __execute__ (calls_len: felt, calls: Call*) -> (response_len: felt, response: felt){}  
    func isValidSignature (hash: felt, signature_len: felt, signature: felt*) -> (success: felt){}
}

* StarkNet
namespace IAccount {
  struct Call {
    to: felt,
    selector: felt,
    data_len: felt,
    data: felt*
  }
  func __validate__ (calls_len: felt, calls: Call*) -> (success: felt){}
  func __validate_declare__ (class_hash: felt){}
  func __validate_deploy__ (class_hash: felt, ctr_arg_len: felt, ctr_args: felt*, salt: felt){}
  func __execute__ (calls_len: felt, calls: Call*) -> (response_len: felt, response: felt){}
  func isValidSignature (hash: felt, signature_len: felt, signature: felt*) -> (success: felt){}
}
The IAccount interface

namespace IAccount {
  struct Call {
    to: felt,
    selector: felt,
    data_len: felt,
    data: felt
  }

  func __validate__(calls_len: felt, calls: Call*) -> (success: felt){}
  func __validate_declare__(class_hash: felt){}
  func __validate_deploy__(class_hash: felt, ctr_arg_len: felt, ctr_args: felt*, salt: felt){}
  func __execute__(calls_len: felt, calls: Call*) -> (response_len: felt, response: felt){}
  func isValidSignature(hash: felt, signature_len: felt, signature: felt*) -> (success: felt){}
}
The IAccount interface

```csharp
namespace IAccount {
    struct Call {
        to: felt,
        selector: felt,
        data_len: felt,
        data: felt
    }

    func __validate__(calls_len: felt, calls: Call*) -> (success: felt){}
    func __validateDeclare__(class_hash: felt){}
    func __validateDeploy__(class_hash: felt, ctr_arg_len: felt, ctr_args: felt*, salt: felt){}
    func __execute__(calls_len: felt, calls: Call*) -> (response_len: felt, response: felt){}
    func isValidSignature(hash: felt, signature_len: felt, signature: felt*) -> (success: felt){}
}
```

Yeah multicalls!!!
The IAccount interface

namespace IAccount {
    struct Call {
        to: felt,
        selector: felt,
        data_len: felt,
        data: felt*
    }

    func __validate__(calls_len: felt, calls: Call*) -> (success: felt){}
    func __validateDeclare__(class_hash: felt){}
    func __validateDeploy__(class_hash: felt, ctr_arg_len: felt, ctr_args: felt*, salt: felt){}
    func __execute__(calls_len: felt, calls: Call*) -> (response_len: felt, response: felt){}
    func isValidSignature(hash: felt, signature_len: felt, signature: felt*) -> (success: felt){}
}
namespace IAccount {

struct Call {
    to: felt,
    selector: felt,
    data_len: felt,
    data: felt*
}

func __validate__(calls_len: felt, calls: Call*) -> (success: felt){}
func __validate DECLARE__(class_hash: felt){}
func __validate_deploy__(class_hash: felt, ctr_arg_len: felt, ctr_args: felt*, salt: felt){}
func __execute__(calls_len: felt, calls: Call*) -> (response_len: felt, response: felt){}
func isValidSignature(hash: felt, signature_len: felt, signature: felt*) -> (success: felt){}
}
Account Abstraction for devs

1. Accounts are smart contracts and must be deployed

2. The address of the Account is computed like a smart contract (not derived from the Signer)

3. Transactions can have multiple signatures
   \[ \text{sig} = \{\text{sig}_1, \text{sig}_2, \ldots, \text{sig}_N\} \]

4. Off-chain signatures (e.g. EIP712) must be validated on-chain by the Account
   \[ \text{You cannot use ecrecover(m, sig) locally!} \]
   \[ \text{You must use account.is_valid_signature(m, sig)} \]

5. You can (and should!) use multicalls
Account Abstraction for devs

1. Accounts are smart-contracts and must be deployed
2. The address of the Account is computed like a smart-contract (not derived from the Signer)
3. Transactions can have multiple signatures
   → sig = [sig1, sig2, …, sigN]
4. Off-chain signatures (e.g. EIP712) must be validated on-chain by the Account
   → You cannot use ecrecover(m, sig) locally!
   → You must use account.is_valid_signature(m, sig)
5. You can (and should!) use multicalls

All you need to remember!
Argent X, first Wallet with native Account Abstraction!

- First Wallet on StarkNet
- Browser extension (Chrome/Firefox)
- Multi-account / multi StarkNet network
- Send and receive tokens/NFTs
- Interact with dapps
- 100% Open source

250k+ downloads

>90% of all StarkNet funds
How Argent X works

Argent X

Account Contract

StarkNet
How Argent X works

Argent X

Do Something

sendTransaction(doSomething)

execute(doSomething)

Account Contract

func __execute__():

doSomething()

StarkNet

Dapp Contract

func doSomething():

doSomething()
Unique Feature: Multicalls

Do $A + B + C$

sendTransaction([A, B, C])

execute([A, B, C])
Unique Feature: (Social) Recovery
Unique Feature: (Social) Recovery

Argent X

StarkNet
Unique Feature: (Social) Recovery

- Time delay of 7 days
- Can be canceled
- 100% non-custodial
- No more seed-phrases!
Unique Feature: Fraud Monitoring / 2Fa

Argent X

StarkNet

Fraud Monitoring Service
Unique Feature: Fraud Monitoring / 2FA

Argent X

Do Something

sendTransaction(doSomething)

Fraud Monitoring Service

Approve? -> confirm with 2FA

StarkNet

execute(doSomething, ...)

func __execute__():

func doSomething():

Unknown Contract

func doSomething():

1. Approve session key with security rules (authorised contracts, max spending, etc)
Unique Feature: Session keys for games

2. Execute game actions without the need to approve on Argent X
Unique Feature: Plugins*

*Collaboration with Ledger and Cartridge
Unique Feature: Use the Secure Enclave

* Uses a different elliptic curve (secp256r1) approved by the NIST
Account Abstraction is needed to scale the UX of self-custody. Let’s make it happen!
Thank you!

Julien Niset
co-founder@Argent
julien@argent.xyz

@jniset
Section 1 details with an image. Enter title here.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Section 4 title here.
Section 4 details with a main point. Enter title here.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in volupatate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Enter your main point / statement here.
Here’s the timeline.

Event 1
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam.

Event 2
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam.

Event 3
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam.
Account Abstraction!

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

- Sollicitudin
- Consectetur
  - Condimentum
    - Magna
    - Ligula

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

- Sollicitudin
- Consectetur
  - Condimentum
    - Magna
    - Ligula
Section 2 details with an image. Enter title here.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.