Stealth Address AA Plugin

TDLR; The proposal is a privacy-preserving smart account utilizing stealth address.

Thanks to @derekchiang and Filipp Makarov for their invaluable feedback.

I. Problem

While Account Abstraction (AA) has been instrumental in providing flexible verification logic beyond the traditional ECDSA signature, there remains the need to confirm your identity and prove ownership of the wallet. Whether it's a private key, an enclave-based key, or even biometric data, some form of identification is essential.

ECDSAValidator

Within the ZeroDev Kernel's ECDSA Validator, the owner's address is stored on-chain. This approach serves the purpose of verifying the signature originating from the signer. However, this could compromise the privacy of smart accounts by making ownership information publicly accessible.

struct ECDSAValidatorStorage { address owner; }

function validateUserOp(UserOperation calldata _userOp, bytes32 _userOpHash, uint256) external payable override returns (ValidationData validationData) { address owner = ecdsaValidatorStorage[_userOp.sender].owner; bytes32 hash = ECDSA.toEthSignedMessageHash(_userOpHash); if (owner == ECDSA.recover(hash, _userOp.signature)) { return ValidationData.wrap(0); } if (owner != ECDSA.recover(_userOpHash, _userOp.signature)) { return SIG_VALIDATION_FAILED; } }

II. Solution

To mitigate this privacy concern, we employs the use of Stealth Addresses. These are designed to obfuscate the identity of smart account owners.

[

S11_eYzg6

2288×896 167 KB

](https://ethresear.ch/uploads/default/original/2X/8/8b386a2b0ed7a321f1f75f171f3caeaf0d59a0ef.png)

Aggregate Signature

However, the practical limitations of generating shared secrets and corresponding private keys within existing wallet UIs present a challenge. To overcome this, we propose using aggregate signatures that can be verified by the contract.

Aggregate ECDSA Signing

Given private key of owner: priv_{owner}

, shared secret key: priv_{shared}

(derived from ephemeral private key and user's public key) and message: m

1. Generate (r,s)

signature from signing m

using priv_{owner}

1. Calculate the aggregate signature s'=s(h+r*priv_{shared})
2. The aggregate signature is (r,s')

Aggregate ECDSA Verifying

The aggregated signature

\begin{aligned} s' &=s(h+rpriv_{shared})\ &=k^{-1}(h+rpriv_{owner})(h+rpriv_{shared})\ &=k^{-1}[h^2+hr(priv_{owner}+priv_{shared})+r^2priv_{owner}priv_{shared}] \end{aligned}

We notice that:

pub_{stealth}=G*(priv_{owner}+priv_{shared})

dh_{owner_shared}=G{priv_{owner}priv_{shared}}

We can thus verify the aggregate signature by:

1. Calculate the inverse of aggregate signature s_1=s'^{-1}
2. Calculate R'

and take its x-coordinate r'=R'.x

R'=(h^2s_1)G+(hrs_1)pubkey_{stealth}+(r^2s)*dh_{owner_shared}

1. The result is r==r'

III. Stealth Smart Account

StealthAddressValidator

Within the framework of our validator, we’ll securely store the stealth address, stealth public key, and the Diffie-Hellman key. Importantly, to bolster user privacy, the owner’s address will not be stored

in the validator. This design ensures that there is no explicit connection between the smart account and its respective owner.

struct StealthAddressValidatorStorage { uint256 stealthPubkey; uint256 dhkey; address stealthAddress; uint8 stealthPubkeyPrefix; uint8 dhkeyPrefix; }

Workflow

The following is the workflow of creating a stealth smart account:

[

Screen Shot 2023-09-28 at 3.02.39 PM

1364×588 25 KB

](https://ethresear.ch/uploads/default/original/2X/9/912e9477117ccfd16620cd8205d4824b13ee0990.png)

Further Improvement

While our stealth smart accounts are self-created, eliminating the need for separate viewing and spending keys used in standard stealth address implementations, our aim is to ensure compatibility with ERC-5564. This will enable senders to transfer tokens to recipients while maintaining the recipient’s anonymity.