ERC-4337 标准研究报告：以太坊为钱包提供的局部最优解

10月 24, 2022 4 13

> 钱包入口从 Web2 以来就是兵家必争之地，当年微信支付春节红包对支付宝的珍珠港奇袭则成为了一段商业佳话。而 Web3 的钱包如今虽然是 Metamask 一家独大，但其频发的安全盗签事故丑闻使得其他竞争者乘隙而入；结合目前行业高度同质化的需求与产品形态，钱包成为了 Web3 创业者高度内卷的赛道。
>
> 区块链上用户账号的终极形态，账户抽象 (Account Abstraction) 一直是以太坊不断在探索的概念，而 ERC-4337 就是由创始人 Vitalik Buterin 带队提出的最为前沿的解决方向。
>
> 跟随着本份研报，让我们一起来看看 Web3 钱包赛道如今的格局、ERC-4337 的实现原理以及其优缺评估。

作者： 十四君，Web3Caff Research 研究员

封面： Photo by ABHAY PADITKAR on Unsplash

字数： 本份研报超 5000 字，预计阅读时长 12 分钟

目录导航

目录

背景

常用术语

文本探讨钱包更深层次的内容，故概述下列常用术语，可跳过阅读。

- AA（Account Abstraction）账户抽象，理想中区块链账户的最终形态。
- EOA（Externally Owned Account）外部账户，目前以太坊兼容链上通过 “私钥或助记词” 所控制的账户，目前以太坊上出现全部交易中，按交易 from 方去重计算总账户数达 1.5 亿，因此 99.9% 都是 EOA。
- CA（Contract Account）合约账户，即以太坊上的智能合约地址，由 to 地址为空的交易触发部署。合约账户地址无私钥，通过交易的 input 可以触发其代码的逻辑。目前以太坊上全地址按 data 字段不为空分析，合约地址约 485W 个。
- SCW/A (Smart Contract Wallet/Account) 智能合约钱包，顾名思义了即合约账户中专门用作钱包的一类型。主流有 Gnosis（多签钱包历久弥新）、Argent（提出社交恢复）和 Blocto（to 个人合约钱包）
- MPC（Multi-Party Computation）多方安全计算。这一种技术方案模式，指分散了控制权形成风控和容灾能力的机制。
- TSS（Threshold Signature Scheme）门限签名是 MPC 的一种实现；即门上有两把锁，需要两个钥匙才能解锁，与多签同理，可以设置 2/3,3/4,3/5 等不同的门限。
- 签名与验签，原理复杂从结果上看，是持有私钥可以对任意内容做签名，其签名结果结合被签名的内容可以恢复出公钥，而公钥的后 20 位即为 EOA 地址，合约中通过验证签名比对地址从而验证权限。

钱包困境何来？

目前钱包账户的安全性是完全依赖于私钥的管理。钱包地址是基于一个随机的私钥选取公钥的一部分。整体的安全头重脚轻；头重部分是基于密码学原理上的私钥不暴露，就不会有人伪造出你的签名。但是如何管理私钥，存储私钥，用私钥执行签名和更换私钥便是整个账户体系最薄弱的环节。

最常见的钱包是如何管理私钥的?

托管钱包中自托管形式是最广为人知的一种模式，Metamask 是目前世界上唯一超千万用户的钱包应用，然而在 2022 年 6 月 21 日，Halborn 研究人员发现了一个问题，在极少数情况下，可以在硬盘上找到未加密的用户私匙（该问题已在 10.11.3 及更高版本的 MetaMask 浏览器扩展钱包中得到修复）。对于浏览器而言，物理访问攻击（他人访问

相关设备）是超出了威胁模型的范畴。因为钱包是建立在浏览器之上，所以要缩减这种攻击面需要耗费大量人力，即便如此也难以完全消除风险。说到底，可能只有全硬盘加密才能为电脑提供强大的抵御物理访问攻击的安全保证。

而目前体验和安全性达到相对平衡的，是 MPC+TSS 的方案。

此方案保持基于密钥管理的环节，密钥从一开始生成多个片段，在最终使用的时候多个片段中各自签名凑到指定数量即可确认，这和智能合约钱包（Gnosis 等）的投票是有着很相似的逻辑。但是这样操作，用户需要管理分片的密钥，甚至有可能需要管理 2 部设备，而且签名的场景可能是很高频的，例如在部分 Gamefi 中一天游玩签名次数多达上十次，这种糟糕的体验将会让玩家对游戏体验造成持续性的厌恶。

除却技术在存储与使用上的局限，另一部分就是源于钱包应用本身、商业化的局限性、安全性能与用户安全感的局限和签名操作内容的欺诈。因此目前钱包整体赛道呈现两路并行的发展，私钥管理应用方面，以及以太坊社区围绕链底层应用层方面，本文将重点放在以太坊标准上的钱包应用发展。

以太坊的账户抽象之路

以太坊的账户抽象之路始于 2016 年由 Vitalik 提出的第一个 AA 模型，其主旨是希望实现一种只要用户支付 Gas 就能为钱包获得安全性的保障。最初的 AA 模型概念是带着强烈的改革机制，因为它依赖于区块链随机数和 ECDSA 加密算法的突破。

在 2020 年以太坊提案引入了 EIP-2938 概念，这是一个同样具有革新意义的提案，因为它会颠覆共识协议（若干底层交易字段，如新的 AA 交易类型，允许不用 ECDSA 和签名也可以调用它，实现验证需要以太坊底层修改签名与验签算法），如果一旦进行就是一件影响力不低于以太坊合并的大事件。

EIP-2938 提出了让合约成为支付费用并开始交易执行的顶级账户（需要注意的是合约钱包仍然是由调用了合约的 EOA 所发起的交易触发的，与 2938 合约即起始账户的想法是不同的）。

图：https://www.numbrs.com/ethereum-erc-4337-proposal/

然而今天的主角，ERC-4337 在 2021 年 9 月由 Vitalik Buterin 作为首发作者提出。

ERC-4337 对比之前的提案，其最大的变化在于类型（Category）的不同。对比下图可以显著地看到，EIP-2938 是 Core 类型而 4337 是 ERC 类型。因为 EIP-2938 需要修改的共识层面修改，甚至需要会引发分叉，而 ERC-4337 只是 ERC 即应用程序级标准和约定，类似于 ERC-20/721/1155 等等，只需要在合约层面定义就可以较为轻松的推出。

图：https://eips.ethereum.org/EIPS/eip-2938

图：https://eips.ethereum.org/EIPS/eip-4337

深入理解 ERC-4337 原理与机制

虽然 ERC-4337 只是 Core 向 ERC 分类的变化，但是 ERC-4337 却承载了实现账户抽象的关键性目标之路，用 EIP 中 Vitalik 的原话来说：" 他能实现帐户抽象的关键目标: 允许用户使用包含任意验证逻辑的智能合同钱包代替 EOA 作为主帐户。"

在 ERC-4337 之前的交易

正如我们一笔寻常的扫码付款，打开微信→扫描商家二维码→输入金额→输入密码→完成支付。

正常的一笔以太坊上转账也有类似的环节和角色操作：

1. 打开钱包：用户管理持有的私钥工具，如 Metamask，Bitkeep，Bitizen 等。
2. 交易签名：用私钥对下列交易必备字段做签名操作比如 to 地址，value 是多少，如果有调用合约则 input 带有调用函数的哈希和入参等。
3. 发送交易：交易被发送到以太坊的任意节点，随后这些节点之间会 p2p 传播，这时交易在待确认交易池（Pending Transaction Pool）里陆续排队。
4. 矿工打包：被选中矿工可任意选择交易池里的交易并打包出块，出块后传播一定区块数后达成最终确定性，从而不再会被回滚，交易在此步骤就被认定上链成功。（这里特别提及一点的是先选定了矿工再由矿工选取交易列表，因此 ta 可以再自己出块的这个块中，由 ta 自己私钥签名额外的交易优先排序。）

以太坊交易字段详解可拓展阅读：

- **当我们在看 Etherscan 的时候, 到底在看什么?** 3.2 章：https://mp.weixin.qq.com/s/Z-TcvaV0Fx0OgaRxZii8sA
- **一种转移并在 Os 拍卖不可转移灵魂绑定代币的方法** 2.4 章：https://mp.weixin.qq.com/s/O7WRJihGewHVmYOQD1uQmg

在 ERC-4337 之后的交易

（请注意，ERC-4337 只是 ERC 层面的规定并不是完全颠覆了账号体系。他的核心变更在于用户发送给矿工的签名内容不同，从交易的签名变为一套合约操作指令的签名。）

我们来重新按完成一笔以太坊转账的流程，来梳理下按照 ERC-4337 实现的交易是什么样的：

1. 打开钱包：用户管理持有的私钥工具，如 Metamask，Bitkeep，Bitizen 等。（不变）
2. 交易签名：用私钥对新的若干字段做签名操作，称之为 `UserOperation` 用户操作对象。（指令不变，但内容字段变化）
3. 发送交易：称之为 bundlers 打包者或是捆绑器，本质仍是由某个负责出块的矿工操作。（发送不变，发送对象改为指定矿工）
4. 矿工打包：`bundlers` 把用户发送的操作签名解析验证后由矿工单独再签名一笔交易来包裹住用户的指令，批量将用户的操作指令转发到某个合约钱包中再由合约来验证用户的签名并执行。（彻底改变）

a. 由于交易是矿工签名并发送的，因此 from 是矿工，原先用户的签名和指令在则在参数之中。

b. 打包发送到作为路由器的智能合约中，执行验证并且进一步转发到各用户独立的合约钱包。

5. 出块流程：完全不变。

两种交易的字段对比

由下表可见基本上的传统交易的参数均在 ERC-4337 的 `callData` 中，而 `UserOperation` 作为整个操作对象不仅有 `callData` 还有其他围绕帮助验证交易和防止 Dos 攻击等设计的参数，比较重要的是 `initCode` 即首次交易时候部署的合约钱包代码。

原交易字段字段	传统交易	在 ERC4337 中
from	即私钥对应的地址	callData 中
to	交易目的地地址	callData 中
value	交易附带金额	callData 中
nonce	交易次数（防重放）	callData 中
inputdata	执行合约用参数	callData 中
gasLimit	交易 gas 费限制	callGasLimit

字段	类型	说明
sender	地址	要操作的合约钱包地址
nonce	数字	防重放参数，也用作创建钱包的盐值
initCode	字节数据	首次交易创建合约钱包的代码
callData	字节数据	包含传统交易全参数
verificationGasLimit	数字	验证交易所需的 gas 量

字段	类型	说明
preVerificationGas	数字	预执行交易所需的 gas 量
maxFeePerGas	数字	即 gasprice
paymasterAndData	字节数据	额外参数，用于填写元交易中付款人的地址
signatur	字节数据	验证环节额外传入钱包的参数

为什么说 ERC-4337 是以太坊给钱包困境的局部最优解？

有了运作原理的基础认知后，我们就可以系统地探讨为什么钱包场景是一个困境？首先从价值开始来讲，钱包可以说是 Web3 入口的第一站，我可以没有资产但不会没有钱包。

- 拥有最多用户的钱包会形成最为有机的认知护城河。

Metamask 超千万的用户为它的产品带来最广泛的行业认可。谁能想到这个最广受应用的钱包插件背后的公司旗下还有两大知名产品，一是 openzeppelin 这个最广的智能合约框架引用库。二是 Infura 这个最大的节点接入服务商。

- 护城河的价值，是给予用户安全之外的安全感。

安全是一个相对的概念，笔者在世界最顶级的互联网公司之一从事安全行业 5 年，见识到国家级科学家们的技术战争，也见识到极为高昂成本的黑灰对抗，安全没有绝对只有相对。而安全感是比安全更难达成的认知传播。

如硬件钱包，其极为的不方便十分容易遭到社会工程学的攻击，但是用户会天然地觉得在我身边的钱包最安全，所以安全感比实际的安全性更为重要。

比如 Metamask，依旧冥顽不灵地允许高风险的 `eth_sign` 来执行签名（此方法用户不可见签名的内容，无法做分析），这导致了每天都有用户被盗签，损失金额可高达百万美金。

- 困境源于人类本身的脆弱性。

管理私钥、助记词、识别签名内容和确定自己签名的每一笔将会发生什么事，这些繁琐的步骤对于而言时间管理成本过高。如果我们回想一下拼多多是如何一步步蚕食淘宝与京东的？无非是由那极简的界面，连购物车都去掉只留下 “拼一拼” 便可以下单等货上门。如此来看 Web2 靠优化体验感去突破 10 亿用户，Web3 却还卡在如何给百万用户做科普上。

- 困境受制于钱包的商业模式。

钱包的本质并不在于沉淀用户资产。钱包只是一个工具的定位，对用户是弱相关性和触达感。

好的工具需要有好的团队持续性的维护和贡献。遗憾的是，目前钱包做不到直接采取收费模式，一旦收费面临他们的只有快速的用户流失，而面向机构的钱包虽然有付费意愿和使用场景却短期内很难形成爆发式的增长。如此来看早期硬件钱包的推出并不只是为了给用户提供安全感，而是为了更好地做出营收。

这些困境在越来越多的钱包开发功能集成后得到了缓解。如今的钱包如同化身为了缝合怪，既有信息流推荐，还有 DeFi 金融服务，甚至提供了 NFT 交易市场数据分析；这乍一看如同目睹了 21 世纪互联网浪潮来临前的门户网站（雅虎等）。互联网历史的前车之鉴来看，如今的钱包入口就是 Web3 用户真正爆发前的入口。

- 技术层的局限性和天花板。

目前来看虽然 MPC 和 TSS 都达成了很大程度上的安全性与便捷性上的兼容与提升，但是无论是 2/2 的弱抗监管性还是 2/3 的用户强自主性优势，托管分片密钥的项目方跑路或遭受攻击也不会影响到用户自己手持的 2 份分片，从而依旧保持着钱包的控制权。

然而用户依旧需要管理这两片分片密钥，一旦丢失将必定面临财产盗窃的危险。

ERC-4337 带来的优点

通过困境可以看到按签名前后具有边界清晰的显著分隔。签名前的私钥管理，目前来看 MPC 和 TSS 实现了最佳的用户体验与安全性审计兼容的融合。而签名后 ERC-4337 则带来了无限的想象空间。基于目前的方案，其优点以及对应原理如下：

- 可以自定义签名算法。

这里的签名只需和合约中签名的算法绑定的（确保合约可以完成解签），而签名这件事本身可以有多种算法实现，不同算法性能和交互模式不同，而这将带来的核心变化是，更好的将签名的功能转入手机设备端实现从而实现便携的硬件钱包。这点主要的挑战是安卓等设备开放性过高，不可能私钥存手机，需要单独的签名芯片等。

- 可以多交易并行。

矿工可以签名聚合多笔交易的数据，只需确保都能执行成功即可。因此交易量足够多就可以拉低平均的 Gas 消耗。

- 可以社交恢复、可更改私钥、可升级及 Non-custodial。

未来的世界将会是 SBT 的世界。如果某天一个人的微信被盗号，即使全部钱被转走，里面的好友列表和聊天记录都可恢复，也依旧能使用原账号因为 SBT 灵魂代币们不会丢失。

既然用户控制的账号本身是智能合约，而签名的对象只不过是对这个合约的操作指令，那么合约本身是可设计升级、可以更改所有权的，账户也自然是无人托管且完全去中心化的。

- 支持元交易和多代币支付 Gas。

对于普通用户而言第一次的上链交互需要先转入手续费再转入要操作的资产，而通过矿工来发送交易（本质是矿工先付了 Gas），用户即可在合约钱包里用其他 Token 的转账来支付矿工 Gas，从而无需先充值原生代币来做手续费。

- 抗 DOS、抗量子攻击。

这个是偏以太坊系统运作层面的设计。如果有人的交易是恶意的，总是在最后一步进行恢复的话，矿工可能会交易执行失败而卡住，所以 ERC-4337 设计不少字段均是用于防止 DOS 攻击的。

如果未来量子计算机出现可以用超快的运算速度来爆破出某地址的私钥，那么基于 ERC-4337 的合约钱包可以使用类似比特币的逻辑在每一笔转账之后都控制合约更改授权指向新的地址，因此就能减少量子计算机计算的时间了（量子计算速度快但没有足够的空间存储全部地址的私钥信息）。

ERC-4337 带来的缺点

- 依然需要管理私钥。

ERC-4337 本质上依旧是依赖于私钥做签名来证明所有权，这点目前除了中心化托管钱包之外暂无解决办法。

- 门槛高、成本高。

合约钱包的门槛相对较高，创建钱包需要花钱。从用户的角度来看，如果项目方不进行补贴就难以有动力去使用产品。

首次交易就要支付创建合约钱包的成本，后续的每次交易操作，按 ERC-4337 的字段的逻辑估算，至少需要 42000 的成本，是普通交易的一倍。

一笔交易的成本是如何计算的呢？主要是发起交易的参数长短和合约计算以及存储的成本。

比如 1kb 的数据存储在链上代价将是 64Wgas，换算成金额则是： `20*(640000)*1e9/1e18 * 2000 =` 25 美金。

可拓展阅读**【源码解读】你买的 NFT 到底是什么？第五章**

https://mp.weixin.qq.com/s/610rn9B2-hg8Bd88W-viXA

- 目前生态产品上对于合约钱包的支持也不足。

比如 DeFi 产品和链桥，由于合约钱包无私钥，自然是无法通过用户签名来证明所有权，需要这些平台二次查询合约标准所有者，然后验证签名挑战。

- 合约钱包也会面临安全挑战因为安全不会有绝对性。
- 多链上支持也受制于矿工，是否支持如此的批量打包验证转发是取决于矿工的意愿。

论未来之路

未来的钱包会何去何从呢？我想用一组数据来说明，截止目前：

- 以太坊总交易中出现的 from 地址去重数为：150682433 约 1.5 亿。
- 合约钱包总数，使用头部两个产品 Gnosis Safe 和 Argent 的数据合计为 15W。

如此来看，这既是千分之一的窘迫，也是千倍空间的机会。智能合约钱包因其可编程型、智能性、复合功能而备受关注，可能成为主流钱包的发展方向。不过辩证地看，它也并不是十全十美的，ERC-4337 还未定为最终的答卷，以太坊数以千计的社区建设者们依旧在寻找更好的破局之道。

免责声明：Web3Caff Research 所发布文章仅代表研究成员及嘉宾个人观点，与 Web3Caff Research 立场无关。本文内容仅用于学习与研究，均不构成任何投资建议及要约，并请您遵守所在国家或地区的相关法律法规。

@0x17: 是的，这篇研报确实质量高。

0 回复

Founder 1 月 前

'是持有私钥可以对任意内容做签名，其签名结果结合被签名的内容可以恢复出公钥'
恢复出公钥，这个是不可能的吧，十四麻烦看看，公钥本来就是公开的，验证者能用公钥解开密哈希，就已经验签并确权了，并不是回复出来的，能恢复密钥那还得了啊😃

0 回复

作者

十四君 1 月 前

> *回复给 Founder*

@Founder: 我实现过基于 eip-712 的签名与验签，和 eip-1271 的核心逻辑是一致的

首先，确实是恢复公钥，但并不是恢复密钥

用户: 对代签内容 X，得出的 y=hash（x)，用私钥对 y 进行签名，得出签名结果

合约:
1: 接受用户上传的 x 和 y，以及签名结果
2: 在合约内的用 ECDSA.recover，恢复出公钥，并截取后 20 位（即地址）
3: 合约再判断该地址是否是预设的地址（如管理员），从而判断签名内容的可靠性

1 回复

相关研报

2022 年度 CeFi 暴雷事件与模式安全性分析万字研究报告：结构性问题与风...

加密行业「挤兑破产」研究报告：从商业银行到 FTX，我们发现了挤兑必然...

去中心化社交网络赛道解构万字研报：全景历史演变、生态现状与未来挑战

GameFi 现有经济模型研究报告：“死亡螺旋”与探寻正外部性

NFT 版税之争研究报告：揭秘版税发展史与 OpenSea 强制站队的黑名单机制

EIP-4337 标准智能钱包实践研究报告：全景式呈现 4337 标准实例实现过程...