干货 | 关于 UTXO 的思考

Ajian | 11. Mar, 2018 | 2689 次阅读

UTXO 账户 引介 Vitalik

什么是 UTXO？

在比特币中，一笔交易“在黑盒子里”实际运作的方式是：*花费*一种东西的集合，这种东西被称为“未被花费的交易输出”（即“UTXO”），这些输出由一个或多个之前的交易所创造，并在其后 *制造* 出一笔或多笔新的 UTXO，可以在未来的交易中花费。每一笔 UTXO 可以被理解为一个“coin（币）”：它有面额、有一个所有者。而且，一笔交易若要有效，必须满足的两个规则是：1）该交易必须包含一个有效的签名，来自它所花费的 UTXO 的拥有者；2）被花费的 UTXO 的总面额必须等于或者大于该交易产生的 UTXO 的总面额。一个用户的余额因此并不是作为一个数字储存起来的；而是用他占有的 UTXO 的总和计算出来的。

如果一个用户想要发送一笔交易，发送 X 个币到一个特定的地址，有时候，他们拥有的 UTXO 的一些子集组合起来面值恰好是 X，在这种情况下，他们可以创造一个交易：花费他们的 UTXO 并创造出一笔新的、价值 X 的 UTXO，由目标地址占有。当这种完美的配对不可能的时候，用户就必须打包其和值 *大于* X 的 UTXO 输入集合，并添加一笔拥有第二个目标地址的 UTXO，称为“零钱输出”，分配剩下的币到一个由他们自己控制的地址。

UTXO 的好处

近来 UTXO 模型已经被推广了，因为它在比特币中的应用，已经一些私有链的用户也在使用它；Hyperledger 切换到 UTXO 的理由如下所示：

> 我们同样正在将我们的账户、余额这样简单化的概念切换到应用比特币 UTXO 模型在事实上的标准，只是稍微作了改进。虽然 Hyperledger 完全不使用比特币，比特币系统仍然是非常强大而富有创造性的，人们已在其中投入上亿美元。通过将比特币的交易模型应用为标准，Hyperledger 的用户将从比特币的创造性中受益；反之亦然，与让 Hyperledger 变得更富互操作性有同样的效果。

除了"比特币的网络效应"，我们可以为 UTXO 模型提出一些技术上的主张；一个特别的主张是：它允许交易的并行化处理，正如一个交易发送者发送两笔独立的交易是，他们可以小心地花费独立的 UTXO ，因此这些交易也可以用任意次序来处理。这种顺序不变性与可并行化属性也许可以带来可扩展性的好处。使一个人的币可以分离开来，同样有一些隐私保护上的好处，尤其是，当一个用户接到的每一笔 UTXO 都使用了一个不同的地址的时候，因为这些地址的私钥可以确切地被所有者通过一个 master seed 生成出来；虽然这种隐私所得很容易被打破，如果该用户并没有仔细地保证他的资金相互分离的话。在本文作者看来，如果隐私是被强烈偏好的，那么由 UTXO 提供的资金分离对于这个任务来说是远远不够的；这将需要更复杂的建构如环签名（Ring Signatures），额外的同态加密（Homomorphic Value Encryption）以及 ZK-SNARKs。

为什么不使用 UTXO？

反对 UTXO 的核心主张有下面两部分：

1. UTXO 的复杂性是没有必要的，而其复杂性在实际运行中会比在理论上还要大。
2. UTXO 是无状态的（stateless），因此并不能很好的适用于比资产的发行和保存更加复杂的应用，复杂应用一般来说是有状态的，比如不同类型的智能合约。

来考察第一种主张。考虑一下你会如何实现一个 UTXO 模式下的钱包——尤其是，生成一个发出交易的函数。这一函数不仅要求一个账户的私钥作为输入，还有一些琐碎的数据，比如一个有序的数字，而不是属于该账户的 UTXO 的全集。这一函数还必须接受集合，并确定一个价值大于需要的输出数额的子集作为输入。某些时候，如果存在多个最小的子集，又会产生一些决定要花费哪个子集的复杂任务。

此外，如果一个钱包真的想要从上面提到的， UTXO 的并行化交易处理属性中获益，该钱包必须仔细地分切“变更输出”以至于该钱包总是有多个变更输出可以用作资金的来源；如果一个钱包只控制一个大的变更输出、总是从中抽取出一个小的数额来做下一笔支出，整个事情又变成连续的了。这不是纯粹理论上的问题；大部分的比特币钱包仍然不能使其最优化，与账户和连续数字模型相比，**这在本质上使其 UTXO 的可并行化收益作废。**

在比特币的例子（现实一点来说，任何一个公有链都是）中，交易费用以每千字节计，而 UTXO 选择算法必须额外地小心以最优化每一笔 UTXO 的长期平均交易消耗；这甚至引发了一个拒绝服务漏洞，攻击者可以使用小额的 UTXO（其价值比花费它们需要的边际手续费还要小） 来堵塞一个钱包。撇开这些，每千字节的手续费的存在在 UTXO 选择算法中引入了一些摩擦：可能有这样一种情况， UTXO 的子集 S 足以支付需要的数额 X，但大小为 S 的交易要求一笔交易手续费 F， 而 S 并不足以支付 X+F，那么 S 就需要增加到 S’ ，但然后 S' 大小的交易又要求交易手续费 F' ，要求有 UTXO 的子集 S''，等等。 简而言之，使用账户和连续数字，创造一个钱包只是一个高中级别的问题；然而，使用 UTXO 它就变得很接近于一个本科生研究级别的挑战了。

UTXO 是如何地不契合于有状态的智能合约，也是清楚的：如果需要创建一个拥有多个阶段的合约，比如，必须由多方提供一些形式的输入，一段时间以后这些参与者又必须执行一些额外的操作，最后，作为他们操作的一个函数，该合约支出资金；很难看出如何拿这个模型去适应基本上无状态而只有花费和未花费的对象。然而，在一个基于账户的模型中，事情就简单了：一个人可以确认一个合约具有他所希望的代码，然后，这个合约就可以被其静态地址调用。

可以给出另一个例子，一个有潜在需求的用例是防止资产被盗的技能，通过引入一个存储在一个安全位置的“复原密钥”，你可以在特定时间之内冲你的主账户撤回交易。在一个 UTXO 模型中，即便交易输出所在的一笔更大的 UTXO 可以对它们发往的目标地址施加一定的要求，这还是一个值得作为学术研究论文的挑战。在一个基于账户的模式中，这可以在20分钟的编程时间里通过一个智能合约来实现，合约只要简单地直接实现这个规则就可以了。对特定各方的有限资产所有权（例如：尚未KYC的用户）是另一个例子，它可以用一些复杂的契约来管理，但在一个基于账户模式的智能合约中它不过是一个简单的代码写作练习。值得指出的是，与其说这些是账户模式相对于 UTXO 的好处，不如说是一个有状态的脚本语言的长处；缺乏一种有状态的脚本语言（例如，在 NXT中）,基于账户的抹上同样无法简单地处理这些问题中的任何一个。然而，可静态寻址的对象的概念使得实现有状态系统的逻辑在实践上变得更加简单、对开发者来说也更加友好。

我们能两者都要吗?

在最近的以太坊实现中，我们有了一个显式的协议层概念，关于账户和交易中的连续数字；因此，我们已经为我们的用户做出了抉择，这是我们用来保护账户的模式。在下一次重要发行，Serenity ，我们正在计划一个抽象模式，将这一选择从协议层下沉到 EVM；本质上，每一个用户都将可以为他们自己选择用于保护他们账户的机制。这打开了朝向创造性的大门，比如，可并行化nonce（本质上，这个方案结合了一个带有千位二进制过滤器的nonce，保证nonce是一次性使用的，但允许用户提前使用未来的nonce，允许高达K笔交易以任意顺序处理），甚至允许用户建立基于 UTXO 的方案，如果他们希望的话。

在后续的以太坊版本中，我们想通过分片实现可扩展性，会有一个跨分片异步调用方案，如果一个合约（以太坊术语，及一个由一段代码控制的账户）希望调用另一个分片中的合约，该合约会在它所在的分片中创建一个“收据”，收据可以被另一个分片上的合约通过默克尔树分支来验证。这种收据的概念在本质上融合了一个异步函数调用过程的概念与一个 UTXO 的概念：如果该函数调用问题是价值转移，则函数调用过程在表面上 *就是* 一个 UTXO ——虽然是一个远远更可一般化的版本。因此，一旦所有这些协议变更实现了，以太坊将在多种形式上支持账户模型和一个 UTXO 模型，允许用户从不管哪一个他们认为对给定应用来说最好的模式中获益。

原文链接: https://medium.com/@ConsenSys/thoughts-on-utxo-by-vitalik-buterin-2bb782c67e53
作者: Vitalik Buterin
翻译: 阿剑

你可能还会喜欢:

[以太坊设计原理]
[干货 | 以太坊的工作原理]
[干货 | zkSNARKs（零知识证明）简述]

4 人喜欢

微信扫一扫
分享至朋友圈

A

[Ajian]

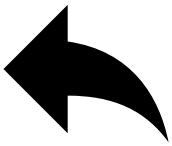
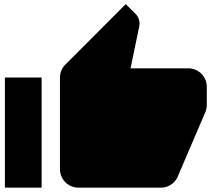
[关注]
[科普 | 对话：无利害关系] [Ajian]
[科普 | 对话：转向权益证明] [Ajian]

- 

  [harveyust]

  Mark``

  7 月前

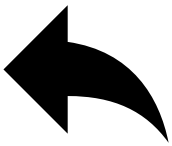
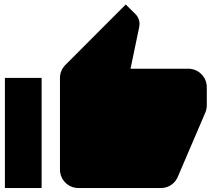
   回复  赞

- 

  [eth1cheng]

  Mark。。

  1 月前

   回复  赞

需要 [登录] 后方可回复

如果你还没有账号请点击这里 [注册]

[RSS] [活跃会员] [帮忙抓虫] [微博] [Twitter] [关于我们 (About Us)]