zkEVM

Note：*从中长期来看，随着 ZK-SNARK 技术的改进，ZK rollup 将在所有解决方案中胜出。* ---- ***Vitalik Buterin***

内容提要

zk-Rollup 是以太坊 Layer 2 扩展性方案皇冠上的明珠：更少的开销且安全等级最高的 Rollup 方案。然而，目前的 zk-Rollup 是限定于一些相对特殊的应用程序，不能迁移现有的 DApp（Solidity 不能直接迁移），很难在 zk-Rollup 中构建通用/可组合的合约。zkEVM，一种运行在 zk-Rollup 上，完全兼容 EVM 且对零知识证明友好的虚拟机，它可以生成用于通用 EVM 验证的 zk 证明，任何现有的以太坊 DApp 和服务都可以平滑地迁移到 zk-Rollup 中。

本文明确了 zkEVM 设计所需要解决的问题以及为什么它现在成为可能，并提出如何从新构建 zkEVM 的具体思路。

背景

zk-Rollup 被公认为以太坊的最佳扩展性解决方案。它与以太坊 Layer 1 一样安全，相较于其他 Layer 2 解决方案，无论计算量有多大，证明都可以非常快速地在链上验证（An Incomplete Guide to Rollups）。

zk-Rollup 的基本思想是将大量的交易聚合到一个 Rollup 区块中，并在链下生成一个简洁证明。然后，Layer 1 的智能合约只需要验证

证明并直接应用更新的状态，而不需要重新执行这些交易。验证证明比重新执行计算开销更少，有效节省了一个数量级的 Gas fee。另外一方面的节省来自数据压缩(即，只保留最少链上的数据用于验证)。

尽管 zk-Rollup 安全高效，但它的应用仍然局限于支付和交易，很难构建通用的 DApp，主要由于以下两个原因：

- 首先，如果想在 zk-Rollup 中开发 DApp，则需要使用 R1CS 编写所有智能合约逻辑。不仅所需语言的语法复杂，而且还需要极强的零知识证明专业知识。
- 其次，当前的 zk-Rollup 不支持可组合性[1]。这意味着不同的 zk-Rollup 应用程序不能在 layer 2 内相互交互。这种特性极大的破坏了 DeFi 应用程序的可组合性。

简而言之，zk-Rollup 目前的功能局限性对开发人员并不友好，我们将通过直接支持原生 EVM 验证，并支持 layer 2 的可组合性，为开发者来提供最佳开发体验。现有的以太坊 DApp 可以平滑地迁移到 zk-Rollup 上。

在 zk-Rollup 中构建通用 DApp

目前有两种方法在 zk-Rollup 中构建通用 DApp

- 为不同的 DApp 构建专用电路（“ASIC”）
- 为智能合约执行构建通用的“EVM”电路

Note: “电路”是指零知识证明中使用的程序。例如：证明 hash(x) = y，则需要使用电路形式重新编写哈希函数。电路形式只支持非常有限的

表达式（R1CS 只支持加法和乘法）。因此，使用电路语言编写程序非常困难，必须使用加法和乘法构建所有程序逻辑（包括 if else、loop 等）。

第一种方法也是最传统的零知识证明方法，要求开发人员为不同的 DApp 设计专用的“ASIC”电路。虽然通过定制电路设计，会减少 DApp 的开销，但由于电路是“静态的”，无法处理动态逻辑，缺乏可组合性。同时电路设计需要很强的专业知识[2]，对于开发者来讲并不友好。

第二种方法不需要任何特殊的设计或者开发者的专业知识。这种 high-level 的想法源于机器的证明：任何程序最终都会运行在 CPU 上。在 zkEVM 中，可以将程序看作智能合约，CPU 是 EVM，因此只需要构建一个通用的 CPU 电路来验证任何程序的执行。但是，这种方法在过去几年中，由于开销巨大并不普遍采用。例如，即使只想在一个步骤中证明 add 之后的结果是正确的，仍然需要负担整个 EVM 电路的开销。假如执行轨迹包含数千个步骤，则证明方的 EVM 电路开销将增加 1000 倍 [3]。

最近，很多的研究按照这两种方法优化 zk 证明，包括：

（1）提出新的 zk 友好原语，像 Poseidon Hash 在电路中可以达到比 SHA256 高 100 倍的效率；

（2）TinyRAM 正在进行提高通用可验证虚拟机效率的工作；

（3）像 Plookup 一样越来越多的通用优化技巧，以及更通用高效的密码库。

Scroll 之前的文章，提出为每个 DApp 设计“ASIC”电路的思想，并且可以通过加密承诺进行通信。经过社区的讨论和反馈后，我们改变了优先级，优先专注于构建通用 EVM 电路(即：所谓的“zkEVM”)，通过设计定制 EVM 电路来解决效率问题。同时不会将设计复杂性留给开发者，zkEVM 将提供与 Layer 1 完全相同的开发体验。

zkEVM 的设计挑战

zkEVM 的构建非常困难。尽管多年来 zkEVM 的设计思路比较清楚，但没有人成功构建原生 EVM 电路。不同于 TinyRAM，zkEVM 的设计和实现更加具有挑战性，原因如下:

1、EVM 对椭圆曲线的支持有限

目前，EVM 仅支持 BN254 椭圆曲线配对。由于不直接支持 cyclic elliptic curve，因此递归证明非常困难。在这种设置下使用其他专用协议也很困难。所以验证算法必须是对于 EVM 友好的。

2、EVM 的单位被定义为 256 位的“字”

EVM 在 256 位整数上运算（就像大多数常规 VM 在 32-64 位整数上运算一样），zk 证明则是在质数域运算。因此，在电路中进行“mismatched field arithmetic”需要大量使用范围证明（range proof），而范围证明是很消耗成本的。

3、EVM 有许多特殊的 opocde

EVM 与传统 VM 不同，它有许多特殊的 opocde（opcode），如 CALL，也有与执行上下文查询和 Gas 相关的错误类型。这将给电路

设计带来新的挑战。

4、EVM 是一个基于堆栈的虚拟机

SyncVM (zksync）和 Cario (starkware)架构在基于寄存器的模型中定义了自己的 IR/AIR。 构建专门的编译器将智能合约代码编译成对于 zk 友好的 IR。这种方法只是语言兼容，而不是原生 EVM 兼容。基于堆栈的模型并且直接支持原生工具链更难证明。

5、以太坊存储方式的巨大开销

以太坊存储方式高度依赖 Keccak 和状态树 MPT[4]，这对于 zk 不是友好的，并伴随产生巨大的证明开销。例如，虽然在电路中，Keccak Hash 比 Poseidon Hash 大 1000 倍。但是，如果将 Keccak 替换为另一个哈希值，则会对现有的以太坊基础架构产生一些兼容性问题。

6、基于机器的证明有巨大的开销。

即使能够正确地处理上述所有问题，仍然需要找到一种有效的方法将它们组合在一起，从而得到一个完整的 EVM 电路。正如上文提到的，即使是像 add 这样的简单 opocde 也可能影响整个 EVM 电路的开销。

为什么现在成为可能

随着最近两年，研究人员在这方面取得的巨大进步，越来越多的效率问题得到解决，因此，zkEVM 证明的成本问题也得到了解决，最大的技术改进主要是以下几个方面：

1、多项式承诺的使用

在过去几年中，大多数简洁的零知识证明协议都坚持使用 R1CS，并在特定应用程序的可信设置中带有编码 PCP 查询。由于每个约束的系数为 2(bilinear pairing 只允许指数相乘一次)，电路无法进行自定义优化，所以电路的大小通常会增大。使用多项式承诺方案，可以自定义设置约束的系数提升到任何程度，极大提升了电路的灵活性。

2、查找表（lookup tables）和 Customized gadgets 的出现

另一个强大的优化来自查找表的使用。最初是由 Arya 中提出，随后在 Plookup 中对 zk 不友好的原语（即与、异或等按位运算）进行了优化，Customized gadgets 可以高效地进行高度约束。TurboPlonk 和 UltraPlonk 定义了优雅的程序语法，以便更轻松地使用查找表和 Customized gadgets。这对于减少 EVM 电路的开销非常有帮助。

3、更灵活的递归证明逐渐可行

递归证明在过去由于依赖于特殊且配对友好的 cyclic elliptic curves（MNT curve），导致递归证明计算开销巨大。 目前，很多技术在不牺牲效率的前提下，递归证明逐渐变得可行。 例如，Halo 可以避免配对友好曲线的需要，并使用特殊的内积参数来分摊递归成本。Aztec 直接对现有协议进行证明聚合（查找表可以减少 non-native field operation 的开销，从而可以使验证电路更小），极大提高了电路的可扩展性。

4、硬件加速

Scroll 已经为证明者设计了最快的 GPU 和 ASIC/FPGA 加速器，

其中，GPU 验证器比 Filecoin 的实现快大约 5 到 10 倍，这可以大大提高证明者的计算效率。发表的论文：PipeZK: Accelerating Zero-Knowledge Proof with a Pipelined Architecture——已经被顶级计算机会议（ISCA）接受。

如何工作，如何构建

在了解 zkEVM 的设计思路和技术改进之外，还需要更清楚地了解证明什么并找出更具体的架构。后续文章中介绍更多技术细节和比较，并详细的介绍 zkEVM 整个工作流程和一些关键思想。

开发者和用户的工作流程

zkEVM 的工作流程对于开发者和用户都是非常友好的，对于开发者来讲，可以使用任何兼容 EVM 的语言实现智能合约，并将编译好的字节码部署在 Scroll 上；对于用户来讲，可以直接发送交易与已经部署的智能合约进行交互。用户和开发者的体验与在 layer 1 上面体验完全相同，而 Gas 费显著降低，并且在 Scroll 上面交易会即刻确认（交易只需几分钟即可完成）。

zkEVM 工作流

Layer 1 和 Layer 2 底层处理过程完全不同：

- Layer 1 依赖于智能合约的重新执行
- Layer 2 依赖于 zkEVM 电路的有效性证明

下面会更详细地解释 Layer 1 和 Layer 2 交易情况的不同之处。

在 Layer 1 中，部署智能合约的字节码存储在以太坊区块链上，交易在 P2P 网络中广播。对于每笔交易，每个全节点都需要加载相应的字节码，并在 EVM 上执行交易，同步至相同状态（交易作为输入数据）

在 Layer 2，字节码也存储在以太坊区块链上，用户的行为也将以相同的方式进行。交易从链下发送到一个中心化的 zkEVM 节点，然后，zkEVM 将生成一个简洁的证明，不仅仅执行字节码，而是证明在交易后状态被正确更新。 最后，Layer 1 的合约将验证证明并更新状态，无需重新执行交易。

接下来深入了解执行过程，从而明确 zkEVM 最终需要证明什么。EVM 执行的时候会加载字节码，并从头开始逐个执行字节码中的 opocde。每个 opocde 都可以被认为执行以下三个子步骤：

（1）从堆栈、内存或存储中读取元素

（2）对这些元素执行一些计算

（3）将结果写回堆栈、内存或存储 [5] 。

例如，addopocde 需要从堆栈中读取两个元素，将它们相加并将结果写回堆栈。

显然，zkEVM 的证明需要包含以下几个方面对应的执行过程：

- 字节码从存储器中正确加载（运行从给定地址加载的正确 opocde）
- 字节码中的 opocde 依次执行（字节码按顺序执行，不丢失或跳过任何 opocde）
- 每个 opocde 都正确执行（每个 opocde 中的三个子步骤都正确执

行读写和计算）

zkEVM 设计亮点

在设计 zkEVM 架构时，我们需要逐一处理/解决以下三个方面：

1、设计 cryptographic accumulator 电路

类似于一个“可验证存储”，cryptographic accumulator 可以有效证明读取的正确性[6]。以默克尔树为例，部署的字节码将作为叶子存储在 Merkle 树中。然后，验证器使用简洁的证明（即：验证电路中的默克尔路径）来验证从给定地址正确加载的字节码。对于以太坊存储，电路只需要与 Merkle Patricia Trie 和 Keccak Hash 保持键值映射关系即可，具体存储可以使用不同的方式实现。

2、设计可以将字节码与实际的执行轨迹关联的电路

字节码中很难移到静态电路中，由于电路中运行字节码之前每个 opocde 都可以跳转到任何其他 opocde，比如：字节码中的条件 opocde，jump（对应于智能合约中的 loop，if else 语句），它可以跳转到任何地方。因此需要一种方法来跟踪电路中的这些跳转，也就是需要验证真实执行轨迹。执行轨迹可以被认为是“unrolled bytecode”，它将包含实际执行顺序中的 opocde 序列（即，如果您跳转到另一个位置，执行轨迹将包含目标 opocde 和位置）。

证明方将直接提供执行轨迹作为电路的见证。强制让程序计数器的值与 opocde 的索引保持一致，所以证明方提供的执行轨迹就是从具有特定输入的字节码中“unrolled ”，另外，为了解决未确定的跳

转目的地，证明者需要通过按顺序提供全部 opocdes，并且证明者不能作弊，可以查找表检查 EVM 证明中的每个索引和 opocde，从而有效地检查一致性(即,证明带有正确全局计数器的 opocde 包含在“Bus”中）。

3、为每个 opocde 设计电路（证明每个 opocde 中的读、写和计算都是正确的）

为每个 opocde 设计电路是最重要的部分（即：证明执行轨迹中的每个 opocde 都是正确和一致）。但是直接把所有的 opocde 聚合到一起，会带来巨大的开销。下面提出几种优化策略:

（1）将读/写和计算拆成两个证明

读/写是将所有 opocde 里面需要的元素提取到“Bus”中，然后证明来自“Bus”元素的执行计算是正确的。这样极大减少了每个部分的开销（不需要在计算证明的时候考虑整个 EVM 存储）。

简显起见，我们将证明分为两个部分。

1）State proof: 证明 Bus Mapping 信息的一致性和正确性。一致性指的 Bus Mapping 和 State 之间的读写一致。正确性指的是 Bus Mapping 中的读写状态正确。

2）EVM proof: 证明 EVM 的 opocde 被正确执行（即证明计算逻辑的正确性）。

（2）每个 opocde 可以自定义约束

每个操作码自定义约束不必使用标准约束,从而可以定义更高的约束（通过将 EVM 字段切成若干块进行有效求解，）。因此，根据

需要通过选择多项式来选择是否“open”约束，可以避免整个 EVM 电路在每一步的开销。

zkEVM 架构最初由以太坊基金会指定，目前处于早期阶段，正在积极开发中。Scroll 正与以太坊基金会密切合作，寻找实现 EVM 电路的最佳方法。到目前为止，已经定义了最重要的特征，并且一些 opocde 已经实现(zkevm-circuits)。更多细节将在后续文章中介绍，建议有兴趣的读者阅读 document，开发进程是实时更新。zkEVM 架构设计将是整个社区共同努力且完全开源，希望更多的人能加入进来并为此贡献力量。

zkEVM 还能带来什么

zkEVM 不仅仅是对以太坊 Layer 2 进行扩展，并且还可以直接通过 Layer 1 的有效性证明来扩展以太坊 Layer 1。这意味着无需任何特殊的 Layer 2 即可扩展现有的 Layer 1。例如，可以将 zkEVM 作为完整节点，可用于直接证明现有状态之间的转换，也就是说不需要将其他内容搬到 Layer 2，即可直接证明所有 Layer 1 的交易。zkEVM 可以为整个以太坊（像：轻量级区块链项目：Mina）一样生成简洁的证明。唯一需要的工作就是递归证明，将完整的递归证明嵌入 zkEVM 电路中是非常重要的，目前最佳递归方式仍然是使用 cyclic elliptic curves（Pasta curve）[7]。

总结

zkEVM 可以为开发者和用户提供相同的体验，并且在不牺牲安全性的情况下开销降低几个数量级。Scroll 目前采用模块化方式构建 zkEVNM 的架构，利用零知识证明方面的最新突破来减少开销（包括多项式承诺、查找表、递归证明和硬件加速）。我们期待看到更多人加入 zkEVM 社区并与我们一起集思广益！

关于我们

Scroll Tech 是一家新创办的技术驱动型公司。我们旨在构建一个具有强大证明网络，且与 EVM 兼容的 zk-Rollup（参阅此处）。 整个团队现在都专注于开发。我们正在积极招聘更多开发人员，请通过 hr@scroll.tech 与我们联系。如果您对技术内容有任何疑问，请通过 yezhang@scroll.tech 联系。

原文链接：https://hackmd.io/@yezhang/S1_KMMbGt

附注

[1]: Starkware claims to achieve composability a few days ago (reference here)

[2]: Circuit is fixed and static. For example, you can’t use variable upper bound loop when implementing a program as a circuit. The upper bound has to be fixed to its maximum value. It can’t deal with dynamic logic.

[3]: To make it more clear, We elaborate about the cost of EVM circuit here. As we described earlier, circuit is fixed and static. So, EVM circuit needs to contain all possible logic (10000x larger than pure add). That means even if you only want to prove for add, you still need to afford the overhead of all possible logics in the EVM circuit. It will 10000x amplify the cost. In the execution trace, you have a sequence of opcodes to prove and each opcode will have such a large overhead.

[4]: EVM itself is not tightly bound to the Merkle Patricia tree. MPT is just how Ethereum states are stored for now. A different one can easily be plugged in (i.e., the current proposal to replace MPT with Verkle trees).

[5]: This is a highly simplified abstraction. Technically, the list of "EVM state" is longer including PC, gas remaining, call stack (all of the above plus address and staticness per call in the stack), a set of logs, and transaction-scoped variables (warm storage slots, refunds, self-destructs). Composability can be supported directly with additional identifier for different call context.

[6]: We use accumulator for storage since the storage is huge. For memory and stack, one can use editable Plookup ("RAM" can be implemented efficiently in this way).

[7]: It's non-trivial to add a complete recursive proof to the zkEVM circuit. The best way to do recursion is still using cyclic elliptic curves (i.e., Pasta

curve). Need some “wrapping” process to make it verifiable on Ethereum Layer 1.