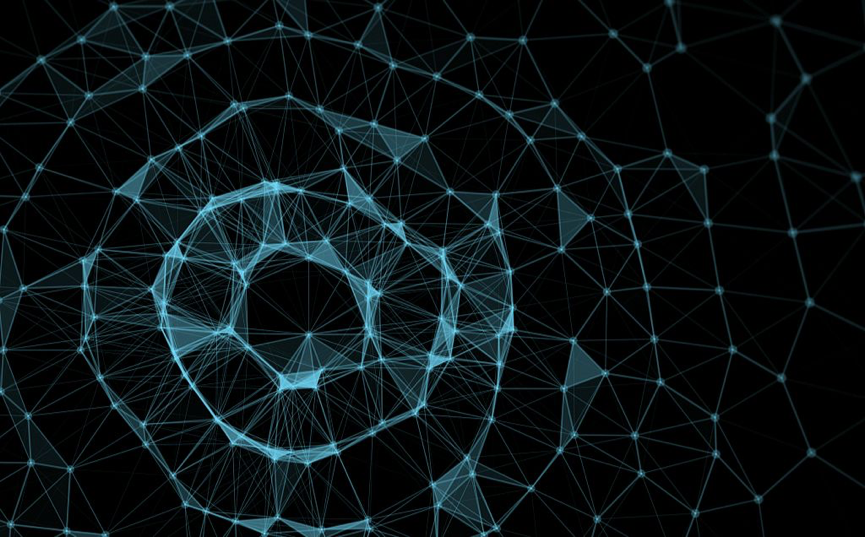
最新以太坊｜零知识证明专题系列报告（一）： 以太坊整合Zcash技术的最新进展（ZoE）

作者：[Christian Reitwiessner](https://blog.ethereum.org/author/christian_r/)



以太坊研发团队的成员和Zcash公司正在一个研究项目上展开合作，该项目旨在解决区块链可编程性和隐私保护功能结合的难题。这篇联合文章由Ariel Gabizon（Zcash）和Christian Reitwiessner（以太坊）合作完成，并同步发布在两家博客上。

以太坊灵活的智能合约界面可以实现各种各样的应用，尽管还有相当一部分的应用我们暂时还没想到。不过依靠这种灵活性，我们添加隐私保护功能的可能性将大大提高。举个例子，我们可以通过智能合约在区块链上进行选举或拍卖活动，活动的结果可以由区块链上的任意观察者验证，但是单个投票或投标的内容不会被揭露。此外，另一种可能的应用场景也许涉及到选择性公开——在该场景中，用户能够证明自己身在某个特定的城市而无须公开自身确切位置。而在以太坊中实现这些功能的关键恰恰是Zcash底层的加密引擎——零知识证明——一种简洁的非交互式知识论证（zk-SNARKs）。

Zcash公司的目标之一（代号“炼金术项目”），是要在以太坊和Zcash之间实现直接去中心化交易。以太坊和Zcash，一个侧重于可编程性，另一个侧重于隐私保护，如果把两者的区块链和技术连接起来，自然而然为开发二者兼具的应用扫清障碍。



作为Zcash/Ethereum技术合作内容的一部分，Zcash的Ariel Gabizon几周前在位于柏林的以太坊中心造访了Christian  Reitwiessner。本次访问的焦点是一个证明——在为以太坊C++客户端实现预编译以太坊合约的基础上，实现用Solidity语言编写zk-SNARK验证者的想法。这项工作补充了Baby ZoE，并为Parity（以太坊的Rust客户端）编写了zk-SNARK预编译合约。我们所做的更新包括添加细微的加密原语（椭圆曲线乘法，加法和配对）以及用solidity语言实现余下功能——这些工作都将使系统拥有更强大的灵活性，并在无须硬分叉的情况下，允许系统使用各种zk-SNARK结构。在我们获得具体实现办法以后，我们会继续跟大家分享办法的细节。我们通过在以太坊区块链的测试网络上验证真正的具有隐私保护功能的Zcash交易来测试新代码，并取得了成功。

验证过程仅花费了42毫秒。该结果表明，把预编译合约添加进系统不仅没有问题，我们还可以把使用这些合约时产生的gas成本调节到一个令人乐于接受的范畴。

**这样的系统可以做什么**

Zcash系统可以在以太坊上重复使用——用于创造具有隐蔽性的自制代币。这种代币早已获得广泛应用，例如投票（见下文）或简单的秘密竞价，也就是参与者在不知道其他人的出价的情况下进行竞价。

如果你想要尝试编译这个概念的证明，可以使用以下命令。如果你需要帮助，请参阅

https://gitter.im/ethereum/privacy-tech



git clone https:*//github.com/scipr-lab/libsnark.git*

cd libsnark

sudo PREFIX=/usr/local make NO\_PROCPS=1 NO\_GTEST=1 NO\_DOCS=1 \

CURVE=ALT\_BN128 \

   FEATUREFLAGS="-DBINARY\_OUTPUT=1 -DMONTGOMERY\_OUTPUT=1 \

-DNO\_PT\_COMPRESSION=1" \

   lib install

cd ..

git clone --recursive -b snark https:*//github.com/ethereum/cpp-ethereum.git*

cd cpp-ethereum

./scripts/install\_deps.sh && cmake . -DEVMJIT=0 -DETHASHCL=0 && make eth

cd ..

git clone --recursive -b snarks https:*//github.com/ethereum/solidity.git*

cd solidity

./scripts/install\_deps.sh && cmake . && make soltest

cd ..

./cpp-ethereum/eth/eth --test -d /tmp/test

*# And on a second terminal:*

./solidity/test/soltest -t "\*/snark" -- --ipcpath   /tmp/test/geth.ipc  --show-messages

接下来，我们还会从多个角度展开讨论将zk-SNARKs整合进以太坊区块链的问题。

**决定有哪些预编译合约需要定义**

回想一下，SNARK是一个针对某些属性的简短证明。为了把隐私保护功能添加进以太坊区块链中，我们要求客户端能够验证这种证明。

在所有最近的结构中，验证程序只包含了在椭圆曲线上的操作。具体而言，就是验证者需要在椭圆曲线组上进行标量乘法和加法运算，还需要进行我们称之为双线性配对的繁重操作。

如上所述，在以太坊虚拟机（EVM）中直接实现这些操作成本过于高昂。因此，我们想要通过预编译合约来执行这些操作。现在，我们辩论的问题是：这种预编译合约应该定位在哪种级别的普遍性（generality）？

SNARK的安全级别分别与曲线参数相互对应。大体上说，曲线次序越大，我们称之为“嵌入度”的参数就会越大，而基于这条曲线的SNARK也越安全。另一方面，这些量越大，对应曲线上的执行操作的成本自然也会越高。因此，使用SNARK的合约设计者可能希望根据它们自己所期望的效率/安全权衡比来选择这些参数。基于这个原因，我们应该实现具有高普遍性的预编译合约。这样的话，合约设计者可以从曲线家族中随意选择所需曲线。我们一开始的确想要实现高度的普遍性，在此情况下，曲线的描述是作为合同输入的一部分给出的。在上述情形中，智能合约将能够在任意椭圆曲线组中执行加法操作。

这一方法的复杂性在于如何为每一步操作分配gas成本。你无法访问特定的实现结果，必须且只能根据曲线的描述去评估在最坏的情况下，使用该曲线进行一组操作的成本有多少。一个普遍性略低的方法是从特定曲线家族选取曲线。我们注意到，在使用Barreto-Naehrig（BN）曲线家族时，用户可以根据给定的曲线参数大致估计配对操作的成本是多少，因为所有这类曲线都支持特定类型的最佳Ate配对算法（optimal Ate pairing）。这里我们大概描述了预编译的工作方式以及如何计算gas成本。

在这一次辩论里我们收获良多，但最终还是决定让概念证明“保持简单”——我们选择基于Zcash现有的特定曲线来执行合约。我们通过使用libsnark库中众多函数对应的wrapper函数来实现这一点（Zcash使用的也是这种办法）。

注意，我们也可以在ZCash当前使用的整个SNARK验证函数中使用一个wrapper函数，就像上文提到的Baby ZoE项目中所做的那样，这种方法很简单。然而，明确定义椭圆曲线操作的优点是系统能够使用更多类型的SNARK结构，这些结构让验证者依照先前提及的三种椭圆曲线操作的组合不断工作。

**将Zcash设置重复用于新的匿名代币以及其它应用**

正如你所听说的，使用SNARK需要经过一个复杂的设置阶段去构建系统中所谓的公共参数。每当我们想要将SNARK应用于某个特定电路时，这些公共参数都需要以安全方式重新生成，这一做法严重阻碍了SNARK的可用性。简化这个设置阶段已经成为我们当下必须考虑的重要目标，但迄今还没获得成功。

好消息是，如果有人希望发行代币，且该代币支持隐私保护的交易，那么他/她可以简单地重用已经由Zcash安全生成的公共参数。这些参数之所以能够重复使用，在于用于验证具有隐私保护功能的交易的电路并非固有地与某一区块链或者某种货币绑定。当然，其显式输入之一是一棵Merkle树的根，该树包含货币的所有有效票据。因此，该输入可以根据用户所希望使用的货币发生改变。此外，发行新的代币也十分简单。你可能通过代币完成了许多任务，然而当时并没有察觉到自己使用了代币。比方说，假设我们希望进行匿名选举，以便在两者中选择首选方案。我们可以为投票发行匿名自制代币，并向每一位投票方分发一枚代币。因为没有“挖矿”过程，所以不可能以其他方式生成代币。现在每一位投票人根据他们所想将手中的代币发送到其中一个地址。最终余额较大的地址便是对应的选举结果。

**其他应用**

构建一个不基于代币且支持“选择性披露”功能的系统相当简单。例如，你可以定期把包含你物理位置（为了防止欺骗行为，可能还包含其他人的签名）的加密信息发送到区块链上。如果你在不同的信息中使用不同的钥匙，那么只有在特定时间，你才能以公布钥匙的方式来揭晓你的位置。然而，通过使用zk-SNARKs，你可以证明你在某一个特定的区域，却无需明确地显示你在哪里。在zk-SNARK内，你对你的位置进行解密并检查它是否在该区域内。由于零知识的特性，每个人都可以验证这个检查，但没有人能够检索到你的实际位置。

**未来的工作**

要实现上述功能——在以太坊区块链上创建匿名代币以及验证Zcash交易，将需要用Solidity语言实现Zcash中使用到的其他元素。

要实现第一个功能，我们必须让Zcash网络上的节点来执行任务，例如更新票据承诺树（note commitment  tree）。

对于第二个功能，我们需要用Solidity语言实现Zcash使用的equihash工作量证明算法（一种内存依赖型算法，译者注）。否则，交易本身虽然可以被验证为有效，但我们无法确知其实际有否被整合进Zcash区块链中。

所幸的是，这样的实现方法还是写出来了。但是。其效率仍有待进一步提高，以便日后应用于实际场景中。

鸣谢：我们感谢Sean Bowe给我们提供的技术支持；感谢Sean和Vitalik Buterin为我们提供了有用的评论。最后，感谢Ming Chan为我们编辑本文。

原文来源：<https://blog.ethereum.org/2017/01/19/update-integrating-zcash-ethereum/>