Phantom：一种基于智能合约的使用 zk-SNARKs 的高效隐私协议

Xing Li[1], Yi Zheng[2], Kunxian Xia[1],
Tongcheng Sun[3], and John Beyler[2]

[1] Unita Technology, lixing@unita.tech
[2] Qtum Chain Foundation, zhengyi@qtum.info
[3] Peking University, suntongcheng@pku.edu.cn

February 18, 2020

Abstract. 隐私对于区块链和去中心化应用而言是一项至关重要的问题。当前，有几个区块链以隐私作为主要特征。例如，Zcash 使用 zk-SNARKs 隐藏交易数据，其中地址和金额对公众不可见。zk-SNARK 技术是安全的，并且已经在 Zcash 中稳定运行了多年。但是，它不能支持智能合约，这意味着人们无法在 Zcash 上构建去中心化应用。
为了解决这个问题，Quorum ZSL 和 Nightfall 这两个协议尝试通过智能合约来实现 zk-SNARKs。通过这种方式，这些协议使区块链上具有隐私特征的去中心化应用得以实现。但是，在以太坊虚拟机上进行的实验表明，这些协议的运行会花费大量时间和 gas，这意味着它们不适合日常使用。
在本文中，我们提出了一种基于智能合约的使用 zk-SNARKs 的高效隐私协议。它有助于使一些去中心化应用（例如数字资产，稳定币和支付）变得隐私。该协议在智能合约的 gas 消耗与 zk-SNARK 生成证明的计算复杂度之间取得平衡。此外，它使用链上秘密分发技术将隐私信息存储在区块链上。一笔隐私交易的 gas 消耗大约是 1M，在一台普通的计算机上生成交易所需的时间不到 6 秒。

Keywords: 区块链隐私 · zk-SNARKs · 智能合约

1 介绍

2008 年 9 月，中本聪发布了比特币的白皮书 [1]。2009 年 1 月，比特币主网正式启动，这开辟了加密数字货币的新纪元，也将一种新的技术引入公众视野。2015 年 7 月，以太坊 [2] 启动。嵌入式 EVM（以太坊虚拟机）可以运行图灵完备的智能合约，这带来了第二代区块链技术。随后，大量带有智能合约的区块链（例如 Qtum、Tron、EOS）和通过智能合约实现的去中心化应用（例如 CryptoKitties、MakerDAO、Uniswap）相继涌现。

2016 年，Zcash 协议 [3] 被提出，该协议使用加密技术为区块链提供增强的隐私性。Zcash 中有两种类型的地址：透明地址（taddr）和隐私地址（zaddr）。透明地址发送和接受交易的过程中，交易的地址和相关的金额会被记录在 Zcash 区块链上，和比特币相类似。然而，隐私地址使用 zk-SNARKs 来保护交易数据，其中地址、相关的金额和加密的备忘数据对公众是不可见的。

基于 Zcash 协议的设计，更多的协议和平台被提出以改进以太坊上的隐私情况。Quorum [4] 是一个开源的区块链平台，它结合了以太坊社区的增强功能和创新功能，以满足企业需求。Quorum 的 ZSL（零知识安全层）是一种协议，该协议利用 zk-SNARKs 来使数字资产通过智能合约进行转移，而无需透露有关发送者、接收者和金额的任何信息。根据其文档 [5] 所述，在 Intel Xeon E3-1225 v2 3.2GHz 处理器（4 核）上，一次 JoinSplit 操作耗时 42.6 秒，并且需要近 3GB 的 RAM。对于日常使用而言，时间和资源的消耗过大了。

2018 年 10 月，安永在布拉格的以太坊 Devcon 上介绍了 Nightfall [6]。在 2019 年 5 月 31 日，安永发布了 Nightfall，可以在以太坊主网上使用 zk-SNARK 智能合约隐私地交易 ERC-20 和 ERC-721 代币。Nightfall 使用深度为 33 的默克尔树的 SHA256 哈希函数。ERC-20 代币的交易需要 2,292,000 个 zk-SNARK 约束，并且消耗约 2.7Mgas，这相对较高，因为以太坊中每个区块的 gas 限制是 8M。

从上可以看到，智能合约上现有的隐私协议不够实用。用户必须等待 40 秒以上才能生成交易，并且 gas 的消耗也无法被接受。在相同的安全级别下，我们减小了 zk-SNARK 证明电路的尺寸和 gas 的消耗。与 Nightfall 相似，我们的协议提供了两种资产：透明资产和隐私资产。前者是通用 ERC-20 代币，而后者的设计与 Zcash 相似，但会更简单。通过对隐私资产采用新的实现方法，最终的性能十分优越：一笔交易的 gas 消耗仅有大约 1M，并且在一台普通机器上证明的生成时间不超过 6 秒。

2 预备知识

2.1 术语

- 票据：票据是对数字资产数值的加密表示。它指定金额和接收方地址。而且，它可以由对应于该地址的私钥来花费。
- 承诺：对于每一张票据，都有一个相关的承诺，该承诺可由一个陷门函数来生成。
- 废弃符：陷门函数还会生一个关联的废弃符。对于每张票据，只可能存在一个有效的承诺和废弃符。
- 默克尔树：默克尔树是一颗树，其中每个叶节点记录数据块的哈希值，每个非叶节点记录其子节点的哈希值。
- 增量默克尔树：增量默克尔树是仅可以进行添加的默克尔树，仅支持数据块的插入。固定深度的增量默克尔树被用来存储票据承诺。
- 链上秘密分发：它用于以加密形式将票据的传输部分存储在区块链上。

2.2 Shrubs 默克尔树

Shrubs 默克尔树 [7] 是增量默克尔树的一种变体。相比之下，它不是由树根来表示，而是由一系列子树的根节点表示。这种设计可以允许以 $O(1)$ 的复杂度插入一个承诺，但需付出更复杂的 zk-SNARK 证明为代价，以证明该承诺在树中。

假设一颗 Shrubs 默克尔树的高度是 h。总共用 $h+1$ 个节点（每一深度有一个）用来表示这颗树。我们称这些节点为 Shrub 节点。在每一深度上，Shrub 节点或是唯一的节点，或是最靠右的完美左子树的根节点。通过这种方式，当插入

新的叶节点时，只需切换到一个新的节点或者重新计算哈希值就可以更新一个 Shrub 节点，且只有这一个 Shrub 节点需要被更新。

这里我们以一个高度为 3 的 Shrubs 默克尔树为例，如图1所示。当插入节点 4 时，Shrub 节点为 14，12，8，4。当插入节点 5 时，Shrub 节点变为 14，12，10，4，因为节点 10 成为了完美子树的根。

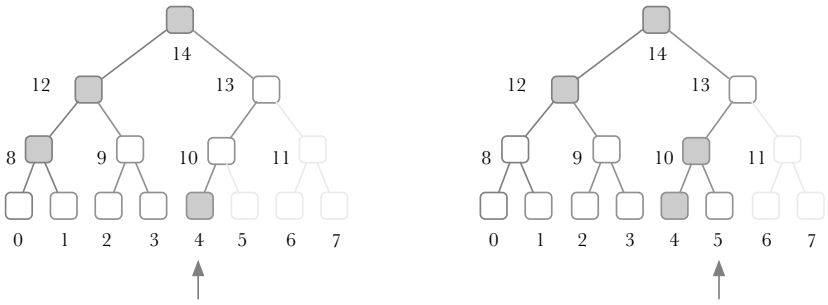

Fig. 1. 一个 Shrubs 默克尔树的例子

因此，数据块插入所消耗的 gas 比原生增量默克尔树小了很多。对于每个非叶节点，它的哈希值最多只会计算一次，当所有叶节点都插入之后，整棵树一共需要计算 $2^h - 1$ 次。所以平均而言，插入一个叶节点只需要进行一次哈希计算。

2.3 哈希函数的选择

为 zk-SNARKs 选择合适的哈希算法时，需要考虑两个因素：gas 的消耗和生成证明的时间。一般来说，一个 gas 消耗更少的哈希函数将导致一个更大的电路(更多约束)，也就意味着更长的证明生成时间。考虑到哈希函数只会在链外进行使用，我们只需关心后一因素。但对于链上使用的哈希函数，则应该平衡好这两个因素。

Table 1. 哈希函数选择

哈希函数	Gas 消耗	zk-SNARK 约束
SHA256	60	28k
MiMC e7r91	8.9k	646
Poseidon t6f8p57	58.4k	317

候选的哈希函数、它们的 gas 消耗（对于 2 个输入）和 zk-SNARK 约束列在了表1中。我们可以看到并没有同时具有低 gas 消耗且小电路尺寸的哈希函数。但是，由于 Shrubs 默克尔树使得构建低 gas 消耗的默克尔树成为可能，因此我们可以在折衷方案中减少 gas 消耗的权重。最后，对于我们的系统来说，Poseidon 或者 MiMC 哈希函数是更好的选择。注意 Poseidon 和 MiMC 哈希函数还没有被大多数密码学专家审核过，因此它们可能会受到攻击。

2.4 Shrubs 公开输入压缩

如上所述，尽管 Shrubs 默克尔树大大降低了将承诺插入树中所消耗的 gas，但它增加了证明承诺在一棵树中所消耗的 gas。基本上，为了证明一个叶节点在 Shrubs 默克尔树上，不仅应该提供从叶节点到其最近的 Shrub 节点的路径，还应该提供所有的 Shrub 节点，后者用于证明到最近节点的有效性。结果是，由于有 $h+1$ 个 Shrub 节点，公开输入的大小由 1 增加到 $h+1$。相关 gas 消耗计算公式如下：

$$\begin{aligned} VerificationGas = \ & n * ScalarMulGas \\ & + PairingBaseGas \\ & + 4 * PairingPerPointGas \end{aligned}$$

其中 n 是公开输入的大小，$ScalarMulGas$、$PairingBaseGas$、$PairingPerPointGas$ 是三个椭圆曲线操作的 gas 消耗。这些操作是通过 EVM 上的预编译合约实现的，它们所消耗的 gas 如表2所示。我们可以看到，公开输入的大小每增加 1，gas 消耗会增加 40,000。因此，Shrubs 默克尔树比原生增量默克尔树多耗费 $40{,}000 * h$ gas。

Table 2. EVM 上椭圆曲线操作的 Gas 消耗

操作	Gas 消耗
ScalarMulGas	40,000
PairingBaseGas	100,000
PairingPerPointGas	80,000

我们引入一种方案：Shrubs 公开输入压缩，用来减少公开输入的大小。如图2所示，为所有 Shrub 节点创建另外一个默克尔树，称作节点默克尔树。我们可以使用节点默克尔树中的一条路径，而不是所有的 Shrub 节点，来证明 Shrub 节点的有效性。这样，公开输入的大小将减小为 1。

每当将叶节点插入到 Shrubs 默克尔树中时，都会更新节点默克尔树，因为 Shrub 节点已经更新。这将给插入数据的过程带来额外的 gas 消耗。为了减少这部分消耗，节点默克尔树使用了一种新的方案来计算默克尔树根，如图3所示。一棵节点默克尔树会被分为几个子树，并且这些子树的树根会被保存下来。当一个叶节点更新时，仅需重新计算相应子树，默克尔树根随之更新。此外，选择 MiMC 哈希作为节点默克尔树的哈希函数，以平衡电路尺寸大小和 gas 消耗。

2.5 公开输入压缩

Shrubs 公开输入压缩能正常工作是因为所有的 Shrub 节点是公开的，并且它们的默克尔树根是固定的。但对于其他的公开输入，这种方法可能不再起作用。这些公开的输入必须受一个单独的电路约束。我们使用一种方法：公开输入压缩，来减少公开输入的大小。Christian Reitwiessner 在博客 [8] 中对此进行了介绍。

假设一个电路可以由函数 $F(u,w)$ 来表示，其中 u 表示公开输入，w 表示隐私输入。可以将此函数更改为 $F(H, f(u,w) \wedge H(u))$ 的形式，所有公开输入的大小变为 1（所有输入的哈希）。MiMC 哈希在我们的协议中用于公开输入压缩。

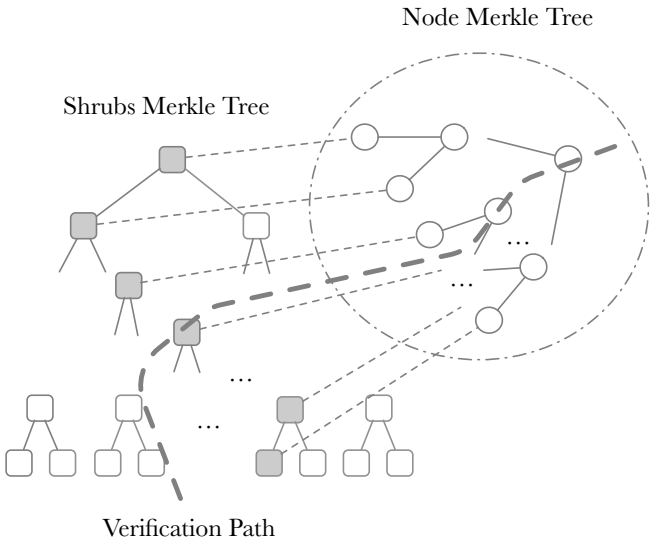

Fig. 2. Shrubs 公开输入封装

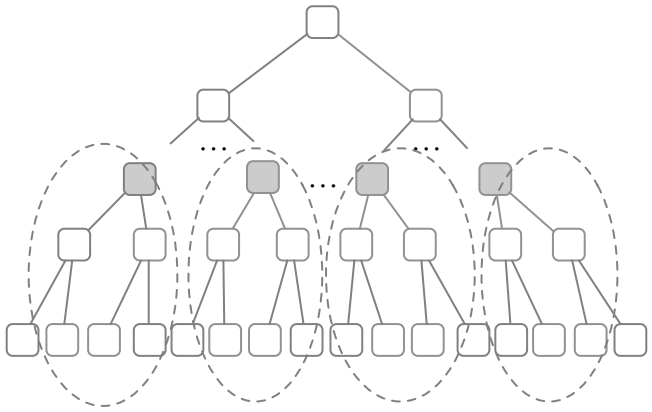

Fig. 3. 节点默克尔树

3 协议

所提出的协议提供了两种资产：透明资产和隐私资产。前者是通用的 ERC-20 代币，后者的设计与 Zcash 类似。隐私资产也可视作 ERC-20 代币的隐私表示。使用这一协议，用户可以轻松地转移他们的透明和隐私资产。

图4说明了协议的总体架构。若干个智能合约部署在了区块链上。Monitor 帮助监控这些合约的交易并将交易发送到区块链上。Server 是核心组件，它调用 zk-SNARK 引擎生成证明并同步区块链的状态。

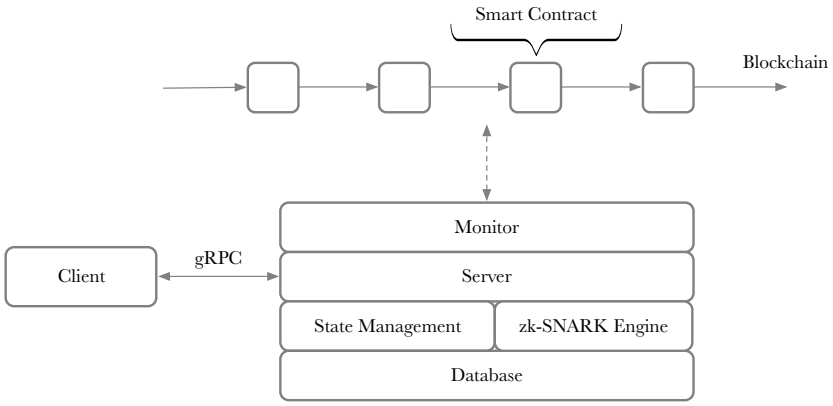

Fig. 4. 协议的整体架构

3.1 密钥管理

两对定义在 F_r 上的公私钥对被用在了协议中。sk、pk 对用于身份识别。esk、epk 对用于链上秘密分发的加密。它们的关系如图5所示。

3.2 票据，承诺，废弃符

协议使用与 Zcash 相同的 UTXO 模型。所有的票据承诺都保存在 Shrubs 默克尔树上。Shrubs 默克尔树的高度被设置为 31，以总共支持 2^{31} 票据。一张票据包含了下列字段

- v：数值，代表数字资产的金额。
- ρ：随机数据，$\rho \in F_r$.
- pk：票据的公钥，$pk \in F_r$.

票据本身对公众来说是隐藏的。而票据承诺会被记录在区块链上，计算方法如下：

$$commitment = commit\_hash(pk, v, \rho)$$

要花费一张票据，应提供相应的废弃符并把它发送到区块链上。废弃符的计算方法如下：

$$nullifier = nf\_hash(sk, \rho)$$

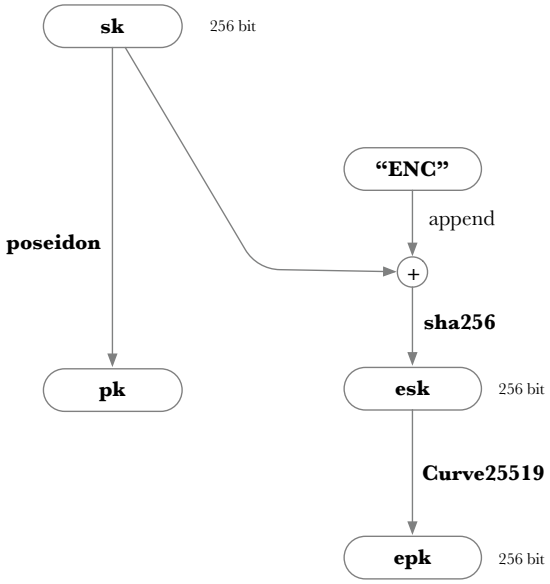

Fig. 5. 密钥管理

3.3 链上秘密分发

在 zk-SNARKs 中，为了传输一张票据，发送方需要通过区块链将承诺发送给接收方，并通过点对点方式将票据值发送给接收方。我们的协议使用链上秘密分发技术，这允许发送方也可以通过区块链秘密地共享票据。这样，整个过程都可以在区块链上完成。

链上秘密分发如图6所示。发送方首先生成一对临时密钥 tsk、tpk。然后通过 tsk 和来自接收方的 epk 生成"共享加密密钥"。最后，发送方使用共享加密密钥加密票据，并将加密的票据和 tpk 发送到区块链上。对于接受方，可以基于 esk 和区块链上的 tpk 恢复共享加密密钥。最后，接受方可以使用共享加密密钥在链上将加密的票据进行解密。

协议中的隐私资产支持三种操作：MINT、TRANSFER 和 BURN。

MINT MINT 操作用于通过将等量的透明资产锁定在合约中，以发行一定数量的隐私资产。如图7所示，它将为已发行资产创建新的票据。zk-SNARK 引擎使用 MINT 电路，以票据的金额和承诺作为公开输入，来帮助生成票据证明。然后将它们发送到智能合约。如果证明被成功验证，则将票据承诺添加到 Shrubs 默克尔树中。

TRANSFER 如图8所示，TRANSFER 操作被用来将隐私资产从发送方转移到接受方。它将生成一个包含四张票据的交易：两张作为输入，两张作为输出。发送方必须为输入提供两个废弃符，为输出提供两个承诺。zk-SNARK 引擎使用 TRANSFER 电路为输出生成证明。然后交易会被发送到智能合约。如果被

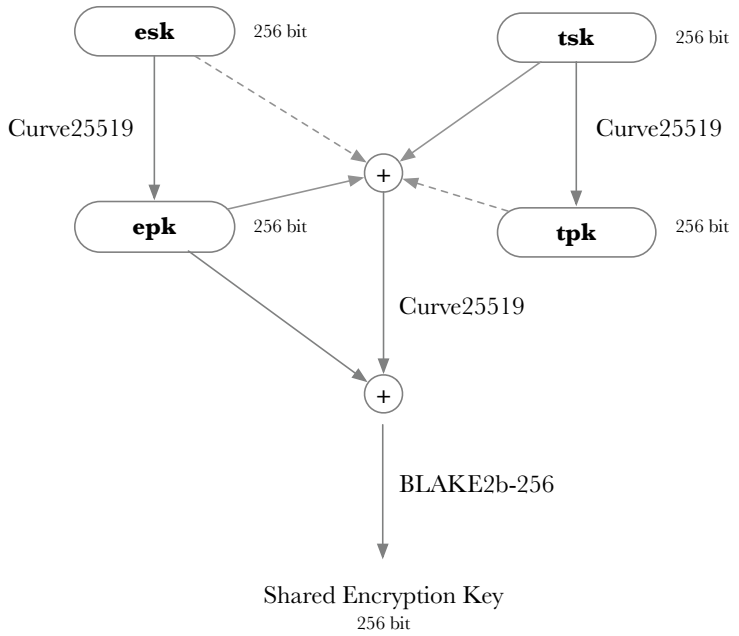

Fig. 6. 链上秘密分发

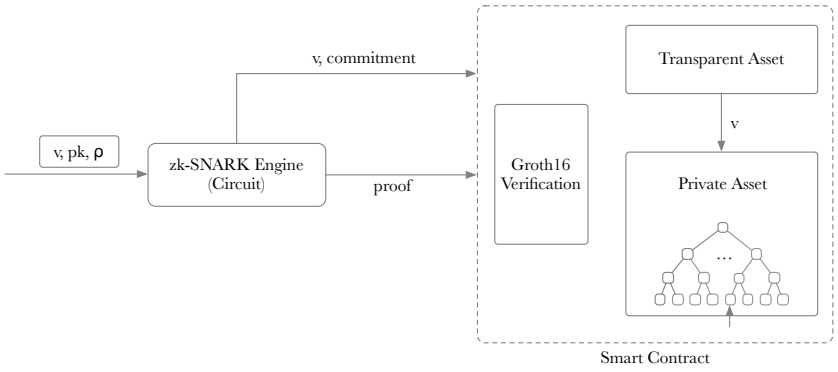

Fig. 7. MINT 操作

成功验证，智能合约将记录两个废弃符，并将两个承诺追加到 Shrubs 默克尔树上。

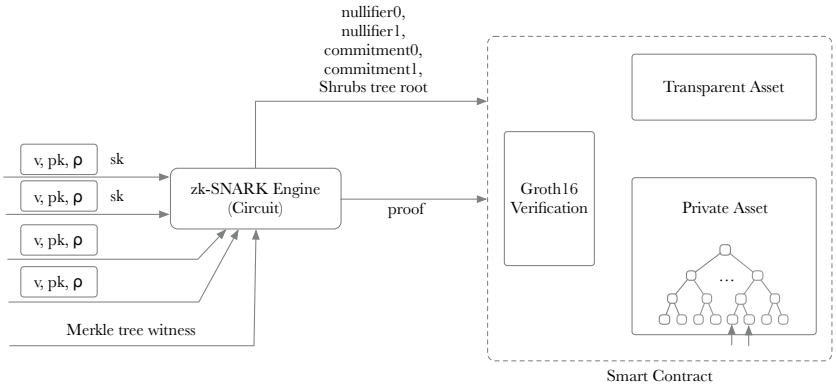

Fig. 8. TRANSFER 操作

BURN BURN 操作可以销毁一定数量的隐私资产并获得等量的透明资产，如图9所示。首先，必须提供票据和其默克尔路径来证明对票据的所有权。zk-SNARK 引擎使用 BURN 电路为票据生成证明。数值、票据废弃符、Shrubs 默克尔树根、账户是发送到智能合约的公开输入。如果被成功验证，票据的废弃符将记录在智能合约中，并且票据也不能被再使用。

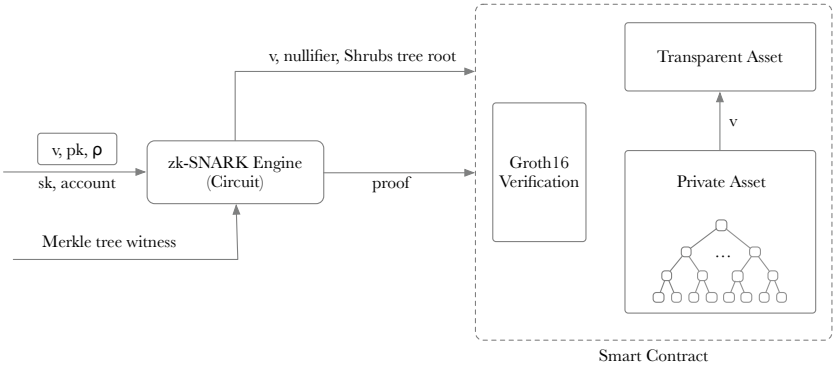

Fig. 9. BURN 操作

4 zk-SNARK 电路

我们用一个例子来描述协议中的 zk-SNARK 电路。Alice 铸造了一定数量的隐私资产，将其转移到 Bob，并由 Bob 进行销毁。下标为 A 的符号表示其属于 Alice，B 为 Bob。

4.1 MINT 电路

MINT 电路被 Alice 用来发行隐私资产。

1. 随机生成一个盐值 ρ。
2. 计算票据承诺 $cm = commit\_hash(pk_A, v, \rho)$。
3. 生成 zk-SNARK 证明 π，该证明断言票据承诺已被正确构建：
   (a) 公开输入 (v, cm)，
   (b) 隐私输入 (pk_A, ρ)，
   (c) 在隐私输入和公开输入上调用 zk-SNARK 证明程序，这将输出一个证明 π，它将检查 $cm = commit\_hash(v, pk_A, \rho)$，并且确保 v 是一个 64 位无符号整数。
4. 生成临时密钥对, tsk, tpk, 并计算共享加密密钥 $EK = BLAKE2b(Curve25519(tsk, epk_A), epk_A)$
5. 生成票据的加密信息 $ciphertext = encryption(EK, (\rho, pk_A))$
6. 将 $\pi, (v, cm), ciphertext$ 发送到智能合约。

4.2 TRANSFER 电路

假设 Alice 想要转移金额为 e 的隐私资产给 Bob。Alice 必须确保自己至少拥有金额总数为 e 的隐私资产承诺。为方便起见，假设 Alice 已经铸造了两个值为 c 和 d 隐私资产的承诺，并且有 $c + d \geq e$。令 f 为平衡量，因此 $c + d = e + f$。

1. 随机生成两个盐值 (ρ_e, ρ_f)，每个输出票据一个。
2. 计算两张输出票据的承诺值：
   (a) $output\_cm[0] = commit\_hash(pk_B, e, \rho_e)$，
   (b) $output\_cm[1] = commit\_hash(pk_A, f, \rho_f)$。
3. 计算两张输入票据的废弃符：
   (a) $input\_nf[0] = nf\_hash(\rho_c, sk_A)$，
   (b) $input\_nf[1] = nf\_hash(\rho_d, sk_A)$。
4. 生成 zk-SNARK 的证明 π，该证明传输操作是有效的：
   (a) 公开输入 $(input\_nf[0], input\_nf[1], output\_cm[0], output\_cm[1], shrubs)$
   (b) 隐私输入：
      i. 旧的输入票据 (pk_A, c, ρ_c) 和 (pk_A, d, ρ_d)。
      ii. 新的输出票据 (pk_B, e, ρ_e) 和 (pk_A, f, ρ_f)。
      iii. Alice 的私钥 sk。
      iv. 对每一个输入票据的默克尔见证 (Shrubs 和节点默克尔树)。
   (c) 在公开输入和隐私输入上调用 zk-SNARK 证明程序，这将输出证明 π，以检查以下约束是否成立：
      i. $output\_cm[i]$ 被正确创建。

ii. $input\_nf[i]$ 被正确创建。
iii. $pk_A = pk\_hash(sk)$。
iv. 对每一个输出票据的默克尔见证是有效的。
v. 金额等式 $c + d = e + f$ 成立。

5. 生成临时密钥对 tsk_A, tpk_A，并计算加密密钥 $EK_A = BLAKE2b(Curve25519(tsk_A, epk_A), epk_A)$
6. 生成临时密钥对 tsk_B, tpk_B，并计算加密密钥 $EK_B = BLAKE2b(Curve25519(tsk_B, epk_B), epk_B)$
7. 生成票据文本的加密文本：$cxt_e = encrypt(EK_B, (e, \rho_e, pk_B))$
8. 生成票据文本的加密文本：$cxt_f = encrypt(EK_A, (f, \rho_f, pk_A))$
9. 将 π、公开输入和两个输出票据的加密文本 cxt_e, cxt_f 发送给智能合约。

4.3 BURN 电路

Bob 可以使用 BURN 操作将他的隐私资产进行销毁。

1. 为输入票据计算废弃符 $nf = nf\_hash(e, \rho_e, sk_B)$。
2. 生成 zk-SNARK 的证明 π，它证明 BURN 操作有效
   (a) 公开输入：nf, e, $account$, $shrubs$。
   (b) 隐私输入：输入票据，sk 和它的默克尔见证 (Shrubs 和节点默克尔见证)。
   (c) 在公开输入和隐私输入上调用 SNARK 证明程序，这将输出证明 ，该证明检查以下约束是否成立：
      i. nf 从输入票据中被正确创建。
      ii. 默克尔见证对于输入票据是有效的。
      iii. 输入票据的值是 e。
      iv. zk-SNARK 证明绑定到公共帐户 $account$。
3. 将 π 和公开输入发送到智能合约。

5 性能

协议在装有 Intel(R) Core(TM) i5-7500 CPU 3.40GHz 处理器(4 核)和 8GBRAM 的计算机上进行了评估。我们使用 Qtum v0.18.1 来启动了一条嵌入了 EVM 的私有链然后把我们的智能合约部署在了上面。每个 MINT、TRANSFER 和 BURN 操作都运行了 20 次来测量平均的证明生成时间，并且我们从区块链数据中取得了它们的 gas 消耗。

Table 3. 协议性能

操作	Gas 消耗	电路约束	电路变量	证明生成时间 (s)
Mint	551k	1892	1896	0.45
TRANSFER	1014k	35323	35328	5.68
BURN	542k	16844	16895	3.05

参数和结果展现在了表3中。我们可以看到 TRANSFER 操作的 gas 消耗大约为 1M，证明生成时间为 5.68s，明显低于其他协议。MINT 和 BURN 操作比 TRANSFER 消耗了更少的资源。整体而言，协议在 EVM 上表现得比其他协议更加高效。

6 总结

本文展示了一种基于智能合约的使用 zk-SNARKs 的高效隐私协议。通过使用一种改进的默克尔树和仔细选择的哈希函数，它将 gas 消耗减小到 1M，并且在常规电脑上的交易生成时间低于 6 秒。此外，它使用链上秘密分发来在区块链上保存私有信息，实现了交易发送者和接受者之间的非交互式交易.

要部署 zkSNARK 电路，需要进行可信设置来生成证明密钥和验证密钥。不幸的是，此过程还会产生一条称为“有毒废物”的数据，必须将其丢弃，因为它可能被用于产生虚假证据，从而破坏系统的安全性。为了解决该问题，可以使用特殊的加密仪式来执行可信设置，在该仪式中，多个参与者轮流执行计算任务，以得到最终的结果。

Zcash 在 2017 年举行了这个仪式。在 2019 年 8 月，Semaphore 团队基于 Zcash Powers of Tau 仪式进行了多方信任设置仪式 [9] 的第一阶段。实际上，仪式可以是永久的，任何 zkSNARK 项目都可以选择仪式的任何一个节点来开始其指定电路的第二阶段设置，并且参加人数没有限制。我们的协议可以从新仪式中受益，并为其生成自己的证明密钥和验证密钥。

References

1. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf`, Dec 2008.
2. Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform. `https://github.com/ethereum/wiki/wiki/White-Paper`, 2014.
3. Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification. `https://github.com/zcash/zips/blob/master/protocol/protocol.pdf`, Oct 2016.
4. J.P. Morgan. Quorum whitepaper. `https://github.com/jpmorganchase/quorum/wiki/`, Nov 2016.
5. J.P. Morgan. Quorum - zsl integration: Proof of concept. `https://github.com/jpmorganchase/zsl-q`, Oct 2017.
6. Chaitanya Konda, Michael Connor, Duncan Westland, Quentin Drouot, and Paul Brody. Nightfall. `https://github.com/EYBlockchain/nightfall/blob/master/doc/whitepaper/nightfall-v1.pdf`, Oct 2018.
7. Marek Olszewski, Eran Tromer, Alexander Vlasov, and Alex Gluchowski. Shrubs - a new gas efficient privacy protocol. `https://devcon.org/agenda?talk=recdu4K0tVPyZahSv`, 2019.
8. Reitwiessner Christian. zksnarks in a nutshell. `https://blog.ethereum.org/2016/12/05/zksnarks-in-a-nutshell`, Dec 2016.
9. Semaphore team. Perpetual powers of tau. `https://github.com/weijiekoh/perpetualpowersoftau`, 2019.