**Solidus：通过PVORM进行的分布式总账交易**

**摘要**

区块链和更普通的分布式分类账作为高效，可靠和持久的数据以及交易记录正在日益受到欢迎。不幸的是，它们通过公开所有数据来确保可靠性和正确性，提出了许多潜在用途的保密问题。

在本文中，我们将介绍Solidus，一个公共区块链上保密交易的协议，例如通过链式结算进行资产转移所需的协议。 Solidus在一个基于现实世界的金融机构的框架下运作：一小部分银行各自拥有大量的用户账户。在这个框架中，Solidus隐藏了交易价值和交易图（即交易实体的身份），同时保持了使得区块链如此吸引人的公开可验证性。为了实现这种强大的保密性，我们引入了可公开可验证的不记名RAM机（PVORM）的概念。我们为PVORM和Solidus提供了一套正式的安全定义，并表明我们的结构是安全的。最后，我们实施Solidus并提出一系列基准，表明该系统在实践中是高效的。

1. **介绍**

基于区块链的加密货币（如比特币）允许用户在可靠的分布式公共分类帐上快速和假名地转移价值。这种在单个分类帐中私人和权威管理资产的能力在加密电子商务以外的许多环境中都很有吸引力。公司已经在分类帐上发行股票[22]，金融机构正在探索基于分类账的系统来实现即时财务结算。

对于这些公司中的许多公司来说，机密性是最大的担忧，比特币类型的系统显然不足。这些系统暴露交易价值和交易实体的化名，往往允许服务和用户的匿名化。对这种泄漏的担忧促使许多金融机构探索解决方案，只有摘要被存储在链上，交易在别处进行。这样的体系结构将区块链减少到仅仅是一个时间戳服务;通过放弃中央化的权威分类账的愿景，他们剥离了许多关键优势的区块链。例如，脱链交易分散在多个数据库中，以便审计和数据保存。

目前区块链的整体结构还不符合现代金融体系的结构。 比特币和类似系统中的直接点对点交易对某些人来说很有吸引力，但却干扰了金融机构的客户服务角色和了解客户的监管要求。 相反，金融业正在探索一个我们称之为银行中介系统的模型。 在这样的系统中，少数实体（我们称之为银行）代表大量用户管理一些链上资产的交易。 例如，一些零售银行可以使用银行中介分类账来权威地记录数百万客户的股票购买。 通过设计，银行中介制度忠实地复制现代金融机构内的资产流动。

虽然之前对银行中介系统的调查很少，但硬币组合方面的工作和最近部署的基于Zerocash的Zcash方案提高了保密性，但是却有明显的局限性。 硬币组合只提供部分保密性。 Zcash依靠零知识的简洁非交互知识（zk-SNARKs）进行匿名，涉及单个消费者机器上报告的证明生成时间超过一分钟。 尽管这样的计算对于执行不经常交易的单个客户来说是可行的，但是对于银行中介系统中的银行而言每秒数百次交易是不可行的。 zk-SNARKs也需要一个不受欢迎的可信任设置，并引入了金融机构不愿接受的工程复杂性和密码硬度假设

为了解决这些问题，我们提供了Solidus，一个支持银行中介分类账的强保密性和高交易率的系统。 Solidus不仅隐藏了交易价值，而且还提供了更具技术挑战性的交易图保密性。这意味着交易的发件人和收件人不能被公开识别，即使是假名。它们可以由各自的银行识别，但其他实体只能了解银行的身份。 Solidus采用了与之前的系统（如Zcash）完全不同的交易图保密方法。作为Solidus的技术基石，我们引入了一个新的基元，称为公开可验证的RAM机器或PVORM，这是一个源于先前在无形RAM（ORAM）上工作的想法。在以前提出的应用程序中，ORAM被单个客户机用来外包存储;只有客户端需要验证ORAM的完整性。在Solidus中，ORAM存储用户账户余额。这意味着系统中的任何实体必须能够验证（以零知识）银行B的ORAM精确地反映涉及的有效交易集合。为了满足这个新的要求，PVORM定义了一组合法的应用程序特定的操作，所有的更新都必须伴有ZK正确的证明。正确性包括账户余额保持非负的要求，每项交易更新单个帐户等等。我们提供了PVORM的一个正式的和一般的定义，并描述了一个包含在Solidus中的实现。  
PVORM的引入为Solidus带来了一些好处。首先，Solidus可以使用基于广义身份证明（GSPs）的有效NIZK证明[14,16]。 GSP更有效地构建zk-SNARK并且不需要可信任的设置，但是验证速度要慢得多，所以我们探索两种选择。其次，与Zcash不同的是，Solidus的核心数据结构只是随着用户账户的数量而增长，而不是随系统生命周期中的事务数量增长。这一特性在高吞吐量系统中尤为重要，并将注入“虚拟”事务以减轻定时副信道的性能损失降至最低。最后，Solidus将所有余额保留为分类帐上的密文。这种方法支持直接的链上结算，例如Zcash不允许的。它还允许授权方解密余额，并允许用户证明自己的余额，例如，如果他们希望将资金从无响应的银行转移出去。

除了PVORM组件之外，我们还以一种理想功能的形式为Solidus整体提供了一个正式的安全模型。作为银行中介分类帐系统的安全要求的说明，这种介绍可能是独立的。我们在这个模型中证明了Solidus的安全性。

此外，虽然Solidus以一个权限分类帐模型为目标，但它只需要一个许可的组;无论是集中还是分散，对基础分类账的实施都是重要的。因此，我们使用通用术语分类账来表示可以以各种各样的方式实例化的区块链衬底。

我们的贡献可以概括如下：

* 银行中介分类帐。我们在Solidus上的工作代表了银行中介分类账的第一次正式处理——一种与现代金融体系中的结算流程紧密结合的新架构。我们的工作提供了一个正式的安全模型，广泛地反映了金融机构将资产迁移到分类帐的要求。
* PVORM。我们引入了可公开验证的无用内存机器，这是一种从ORAM派生而来的新的构造，适用于在分类帐系统中实现事务图机密性。我们使用广义身份证明提供正式的定义和有效的结构。
* 实施和实验。我们报告了我们对Solidus的全面实施，并介绍了基准实验的结果。

我们的结果不仅仅是一个新的技术方法来处理分类账的交易图机密性。他们还展示了银行中介分类帐系统的实用性和全面的链上结算。

1. **背景**

我们现在审查现有的加密货币计划和保密方法。然后，我们解释银行中介系统所建立的金融系统的一些背景，并介绍用于在Solidus中实现安全性和机密性的技术构件。

**2.1现有加密货币**

许多流行的加密货币是基于同样的一般性由比特币推广的交易机制。任何用户U都可以使用公共密钥/私人密钥对创建一个账户（比特币“地址”）。 为了转移资金，U通过签署一个请求向接收方发送一定数量的硬币来创建一笔交易T.矿工对交易进行排序，并直接将T发布到区块链，即交易的权威附加记录。 由于只记录交易，为了确定U的余额，有必要在整个区块链中统计涉及U的所有交易。 作为一个性能优化，许多实体维持一个平衡总结 - 称为比特币中的未支付交易（UTXO）。

这种设置公布了所有账户余额和交易详情。 唯一的保密来源于公共密钥的化名，这些公钥很难 - 尽管不可能与现实世界的身份联系起来。

为了隐藏余额和交易价值，麦克斯韦提出了一个名为“机密交易”（CT）的方案。 CT采用类似比特币的模式，但只发布Pedersen对余额的承诺。 交易价值同样隐藏，平衡使用承诺同态进行更新，并使用广义身份证明（见下文）证明非负。 Solidus使用在El Gamal密文上操作的CT变体来隐藏交易值。

Zcash的确提供了很强的保密性。 余额，转账金额和交易图都保持隐藏状态。 不幸的是，它与财务结算系统不一致，如下所述，有几个严重的缺陷。 首先，Zcash中的Merkle树（权威状态）与整个系统的交易历史成线性增长。 在高事务吞吐量的系统中，这将会迅速降低性能。 其次，实际的zk-SNARKs需要一个值得信赖的设置，如果这个设置被破坏，可以证明伪造和盗窃硬币。 最后，在Zcash中生成证据非常昂贵——据报道在单个消费者机器上平均超过一分钟。 这种开销在像Solidus这样的银行中介系统中是被禁止的。 这些缺陷和与金融业需求相一致的愿望激发了我们截然不同的方法。

**2.2银行中介系统**

管理分类帐上的资产对金融业来说是个吸引人的选择。

今天在金融市场上的资产转移涉及到一个艰难的三步过程。执行是指买卖双方交换资产（如现金证券）的法律上可执行的协议。清算正在更新分类账以反映交易结果。结算是指结算后交换资产。多家金融机构通常充当中介机构; 当客户购买证券时，经纪人或银行将通过信息交换机构代为清算。

比特币等现有的加密货币可以被视为数字资产的标题。在包含交易的块被开采时，执行采用数字签名的交易请求的形式，清算和结算同时完成。

然而，今天，银行介入大部分金融交易。 即使使用比特币，普通客户通常也将账户管理延迟到交易所（例如Coinbase）。 此外，一套迷宫般的法规，比如Know-Your-Customer，则倾向于银行间的系统。 因此现有的加密货币与金融行业或普通客户的需求并不一致。

Solidus的目标是在以银行为中介的基于台账的环境中提供快速的交易结算。 与标准加密货币一样，Solidus假定每个用户都有一个公钥/私钥对，并对交易进行数字签名。 然而，Solidus将账户余额和交易金额隐藏为密文。为此，同时提供公开的可验证性，它依赖于PVORM。

**2.3Oblivious RAM**

由于PVORM深受Oblivious RAM（ORAM）的启发，我们在这里提供一些背景知识。

ORAM是一种密码协议，允许客户安全有效地将数据存储在不受信任的服务器上。客户端维护从逻辑内存地址到远程物理地址的映射，并执行远程读取和写入操作。在这种设置下确保数据的新鲜性，完整性和机密性可以通过使用经过身份验证的加密和最低限度的本地状态直接进行。ORAM的关键属性是隐藏存储器访问模式; 多项式有界对抗服务器不能区分两个相同长度的客户端读/写操作序列。

这些属性为Solidus提供了一个吸引人的构建模块。 识别系统交易图中的边缘可以很容易地减少到识别哪个账户的余额随交易而改变。 因此，将所有余额立即放置在ORAM中可以提供交易图的机密性。 而且，近期的工作大大提高了ORAM的性能。 最实际的ORAM结构在客户端维护一个小地方缓存，称为存储，或者将数据块组织成一棵树，允许每次访问时进行对数运算，或写入完全随机的位置，从而产生恒定时间 写入但是线性读取（所谓的“只写”ORAM）。

不幸的是，标准的ORAM对Solidus来说是不够的。 由于ORAM是为使用不可信服务器的客户端设计的，因此正确性意味着ORAM反映了客户端的更新。 没有“有效”更新的概念，更不用说为客户证明更新的有效性了。 在Solidus中，客户（银行）必须证明应用程序定义的每个更新的正确性。 银行也不能存储本地存储，因为我们不再拥有分类帐上的所有数据。 为了解决这些问题，我们在第四部分介绍了PVORM——一个受ORAM启发的新构造。

**2.4广义身份证明**

Solidus大量使用广义身份证明（GSPs），这是一类Σ-协议，可以为其构建实用的真实验证者知识的零知识论证（或证明）。

引入的符号为调用PoK语言的GSP提供了强大的说明语言。如果x∈Zp且y = gx，则PoK（x：y = gx）表示x的知识的ZK证明，使得y = gx其中g和y是验证者已知的。（这是身份识别协议。）

用于GSP的PoK规范语言相当丰富; 它支持任意数量的变量以及谓词之间的连接和分离。 它具有一套基于身份识别协议的相应标准工具，在G已知的情况下可以在实践中高效实现。 另外，使用菲亚特 - 沙米尔启发式可以使GSP不具有交互性，即生成NIZK知识证明。

Solidus以各种方式使用GSP来确保帐户余额，并正确更新PVORM并保持有效。

1. **Solidus概述**

在深入研究技术细节之前，我们将对Solidus进行概述，包括基本符号，信任假设和安全目标。 我们也给一个架构的草图。首先，我们给出一个具体的目标应用作为动机。

例子1 （TradeWind Markets）。 TradeWind Markets的使用案例为Solidus的设计提供了信息，Solidus提供了Solidus如何支持在账本上管理资产所有权的例子。 TradeWind目前正在建设一个电子通讯网络（ECN），用于实体黄金交易，以电子方式使用银行中介账户进行交易设置和所有权管理。实物金条由托管人管理，托管人可以信任追踪流入和流出专门指定的金库。每个用户都有一个控股银行（通常是大型商业银行）的账户，负责管理交易。用户可以额外从外部购买黄金，将其发送到金库，然后在TradeWind ECN上出售，或者在TradeWind ECN上购买黄金，将其从金库中移除，然后在别处销售。在前一种情况下，托管人必须在托管银行创建账户中新资产的记录。在后一种情况下，保管人必须销毁相应的记录。

控股公司在分类账中代表资产单位 - 个人用户持有的黄金分数盎司。 当用户在TradeWind ECN上交易黄金时，她授权其控股银行将一些黄金单位转让给另一个用户。 控股银行也可以根据用户的账户余额提供其他服务，例如持有黄金作为贷款的抵押品。 在这种情况下，银行可能会冻结部分或全部用户的资产，例如，直到贷款被偿还。

正如我们将要展示的那样，Solidus可以支持像TradeWind ECN这样的系统的全部资产生命周期，同时提供实际的性能和强大的机密性和验证性的保证。

* 1. **设计方法**

Solidus有两个重要特征，与现有的分类帐系统不同，使其更适合金融业。

在他们的账户中持有一些资产（例如，证券，数字货币或黄金）的份额。 正如我们在下面讨论的那样，被称为资产公证机构的特别指定实体记录资产注入系统。 其次，Solidus提供了强大的保密性。 它隐藏了来自非交易实体的账户余额和交易细节，将它们作为密文放在分类账上。 正是由于这些原因，Solidus使用了PVORM。 每家银行在账本上都有自己的PVORM记录账户的身份和余额。

每笔交易涉及发送银行的发送用户和接收银行的接收用户。 当一个用户（寄件人）我们签署一个交易并将其交给（发送）银行时，Bs首先验证交易的有效性 - 它正确地签署了，而Us则拥有$ v的资金发送。 然后Bs更新其PVORM，以反映交易的结果，从美元余额中扣除$ v。 接收银行然后对接收用户的账户进行类似的更新，将其余额增加$ v。

PVORM的保密特性确保了另一个实体只能学习发送和接收银行的身份，而不是$ v或者交易用户的身份。 事实上，即使是发送银行也不能识别发送用户的接收用户和接收银行.6 PVORM的公开可验证性确保了任何可以访问分类帐的实体都可以验证两个银行的正确处理。 具体而言，发送银行对PVORM操作的验证检查了三件事：交易由该银行的一个用户有效签署，$ v是从同一用户的余额中减去的，并且用户的结果余额是非负的。 收款银行执行PVORM操作，将$ v添加到接收用户的账户余额，包括正确的余额更新证明。 Solidus中的PVORM证明是使用通用身份证明生成的。

Solidus被设计为不可知的底层分类账的工具。 虽然它确实需要一个由银行账户维护人员对银行和交易进行相互感知的验证，但这些维护者可以是一个“被许可的”（固定实体）组，一个“未被许可的”（完全分散的）分类账（区块链） 其他值得信赖的仅附加数据结构。

* 1. **建筑模型**

在Solidus中，一组预定的银行B1，...，Bm将资产标题保留在分类帐上。 每家银行都有一个用于加密和签名的公钥/私钥对。 它也有多达n个用户，每个用户都有自己的签名密钥对。每个账户都与一家银行有独特和公开的联系，所以银行（U i）= Bi是明确的。每家银行都维护自己的私人数据结构 Mi包含每个用户的余额和公钥。 它维护一个对应的公共数据结构Ci，置于分类账上，其元素在Bi的加密密钥之下被加密。 Mi和Ci一起构成PVORM中的记忆，我们将在第4节中介绍。Solidus使用此结构来确保对Ci的更新反映在Mi中处理的有效事务，同时隐藏交易细节和交易图。

交易T是由用户Ui j进行数字签名的请求，余额$ bij将一些数额$ v的资产发送给另一个用户。交易是有效的如果。为了处理事务，Bi 更新为Mi 来设置和 Bi0更新Mi0来设置。 他们生成公开可验证的$v≥0的ZK证明，并且使用$ v正确地更新了各自的PVORM。 图1描述了一个简单的Solidus交易。

我们将分类账视为验证交易的公共附加存储器。 所有银行都具有异步认证的读写访问权限，分类账仅接受尚未存在的格式良好的交易。我们通过一个理想的功能FLedger来进行建模，详见附录C，任何银行都可以调用。

公证新资产标题。如上所述，全部用户交易必须是零和; Ui寄钱给我（她必须拥有）汇去U j'。虽然这对于用户是有意义的，但金融系统一般不会关闭。也就是说，资产可以通过特定的渠道进入和离开系统。为了支持这一点，Solidus定义了一组固定的资产公证{U1 $，…… ，Ul $}。这些帐户没有记录余额。如果Ui $向U ij发送货币，则不需要检查$ b $ i（不存在）。为了方便审计这个敏感的行为，并避免试图模仿一个没有平衡的用户的定期更新，我们只需显示Ui $的身份。

资产公证显然必须受到限制;允许任意用户创建和破坏资产标题是没有意义的。在示例1中，Solidus将指定保管人作为唯一的公证人，负责确认收取和移除实物资产（黄金）并保证其物理完整性。

* 1. 信任机制

Solidus认为银行尊重他们自己的用户的保密性，否则不需要诚实行事。 他们可能试图窃取金钱，非法向其他银行提供资金，操纵账户余额，伪造证据等。银行（分别为用户）也可能试图被动地或分别违反其他银行用户（分别为其他用户）的保密性积极。 我们假设没有受到损坏的银行或用户数量的限制。

分布式账本。我们假设3.2节给出的分类帐抽象表述。实际上，分类帐可以但不一定由银行自己维护。 如果不是由银行维护，分类账的信任模式与上级协议无关。 它可以使用（Paxos，ZooKeepe或Raft这样的一个（容许碰撞）共识协议，一个拜占庭共识协议如PBFT，一个分散的共识协议 Nakamoto共识，甚至是一个可信赖的实体。 我们简单地假设分类账维护者满足协议的要求，分类账保持正确和可用。

我们将分类账与公共PVORM数据结构{Ci}一起视为复制状态机。 尽管如此，Solidus的灵活设计使我们可以将共识和应用层作为我们大多数讨论的完全独立的对象。

可用性。我们假设分类帐始终保持可用; 它不容易受到拒绝服务攻击，并且有足够的共识节点保持活跃状态仍然没有错误。 但是，银行可能有两种不可启用的方式：它可以通过拒绝交易来冻结用户的资产，也可以完全脱机工作。

资产冻结可以是一个功能。 对于某些类型的资产（例如，如例1中的黄金），用户可能希望将其余额用作贷款抵押品。 在这种情况下，银行必须能够确保资产保留在用户的账户中，直到贷款被偿还。 与此同时，用户可能仍然需要拥有资产的记录。 资产冻结支持这个用例。

但是，由于技术或业务故障，银行可能会恶意冻结用户的资产或下线。在任何一种情况下，具有银行解密密钥的审计师（见下文）都可以使用户证明自己的余额，并且尽管无法直接办理交易，也可以收回资金。

审计。监管机构和审计师在金融部门扮演着举足轻重的角色。 尽管Solidus不包括特定的审计支持，但它确实能够使银行证明正确的在链数据解密或与可信第三方共享私钥解密密钥。 在第一种情况下，审计员可以根据需要获取交易日志并验证日志的准确性和完整性。 在第二种情况下，审计师可以直接并主动监测银行及其账户的活动。

**3.4安全目标**

Solidus旨在为个人用户和整个系统提供非常强大的安全和保密性保证。

安全保证。Solidus提供了一个非常简单但强大的安全保证。 首先，没有用户的明确许可（以签名的形式），用户的平衡可能会减少。 而且，从账户中移除资产的任何授权只能使用一次; 没有重播攻击。 其次，没有用户可以花钱，她没有。 我们通过确保账户余额永不消失来实施这一保证。 最后，不包括资产公证人的交易必须是零和; 发件人的余额总是减少与收件人余额增加相同的金额。

为了确保上述性能，我们要求每一笔交易的正确性以公开可核实的方式证明（通过ZK-Proof，如上所述）。 如果分类账在接受和结算交易之前检查这些证据，那么每笔交易将维持这些担保。 Solidus将所有证据放在分类账上，这意味着他们可以另外由审计员在线下进行验证。

保密性保证。为了便于对恶意银行进行付款和资产追回，Solidus将所有账户余额和交易详细信息直接放在账本上。 尽管持续不断的公开记录，Solidus为所有用户提供了强大的保密性，如上所述。 首先，账户余额在用户银行（以及任何授权阅读该银行数据的审计员）之外是不可见的。 其次，虽然交易方式显示了发送和接收银行，但无法确定涉及同一家银行的两笔交易是否涉及同一账户。 这第二个特征通常被称为事务图机密性。 它不包括使用比特币和类似系统使用的假名方案，也是PVORM特别提出的挑战。

我们不直接处理交易时间泄露的信息。 Zcash和类似的系统也没有解决这个问题，但Solidus的卓越性能更好地支持一系列的副通道对抗措施。 例如，为了消除定时副通道，Solidus可以定期发送批量统一大小的交易，这些交易是通过值为0的“虚拟”交易填充的。

我们在第5部分提出了一个正式的模型，包含所有这些安全和保密目标。

1. **PVORM**

如第2.3节所述，ORAM提供了隐藏Solidus交易图的手段，但缺乏Solidus要求的公开可验证性。 为了克服这个限制，我们引入了可公开验证的遗漏内存机器（PVORM）。

与ORAM一样，PVORM具有私有逻辑内存M和相应的加密物理内存C.然而，有四个关键的区别：

•受限更新。写操作由公共函数f约束。例如，在Solidus中，M包含帐户ID和余额，f将单个余额更新为非负值。

•公共可核实更新。如果客户模式C不能公开验证，则必须公开证明（以零知识）该更改反映了f的有效应用。

•客户端维护所有内存。客户端维护M和维护C的服务器，客户端直接维护。当M仍然存在时，C现在是公开可见的，例如在Solidus的分类账中。

•没有私人藏匿。 M中没有用C表示的任何数据将阻止客户端证明写入的正确性。 PVORM包含一个固定大小的公用加密存储器，而不是可变大小的私有存储器。

为了实现公开的可验证性，我们的PVORM构造依赖于公钥密码学。 传统的ORAM使用对称密钥原语，但这种差异不是根本性的。 人们可以使用对称密钥加密和zk-SNARKs构建一个PVORM，但正如我们在7.3节看到的那样，这样的构造表现不佳。

我们还利用了PVORM为公众可验证而设计的事实，而不是存储外包以提高效率。 在ORAM中，读取会产生成本，因为客户机必须从服务器检索数据。 在PVORM中，读取是“自由的”，因为它们只需要读取公共状态 - Solidus中的分类帐 - 它什么也不漏。 然而，作品仍然是公开可见的。 其次，由于PVORM并不旨在减少本地内存的使用，所以我们假设客户端本地维护PVORM的全部副本，包括私有数据和元数据。 这允许客户端通过避免不必要的解密来更有效地执行更新。

这些功能几乎与只写ORAM的功能相同，但我们不能以相同的方式使用它们。 事实上，我们的构建基于通用ORAM（电路ORAM），因为它允许我们将更新作为读取、更新、写入操作来实现。 如果我们使用基本的写入操作（就像只写入ORAM所要求的那样），我们就无法证明新旧值之间差异的关键特性。

**4.1形式定义**

我们现在提出一个PVORM的正式定义。 我们让M表示来自公开定义的空间（例如N）的私有值的数组，并且C是M的公共（加密）表示，如上所述。 U是更新规范的空间（例如账户ID，余额变化对）。

定义和正确性。我们首先将PVORM的公共接口定义为正确的操作。 PVORM包含以下操作:

•初始化（1λ，n，m0，U）→ - （pk，sk，C）是一个随机函数，用安全参数1λ初始化PVORM，n个数据元素，初始存储器M =（m0，...，m0） 和一组有效的更新值U。

•更新约束函数f（u，M）→Mr根据更新u∈U更新M。 注意f在某些输入上可能是未定义的（无效更新），并且如果u∈/ U则必须是未定义的。

•更新（sk，u，C）→ - $（C'，e，证明），这是一个随机更新函数，用于更新u和公共存储器并发出新的公共存储器，u的密文e，以及正确应用的零知识证明。

•Ver（pk，C，Cr，e，证明）→{true，false}，一个确定性的更新验证函数。

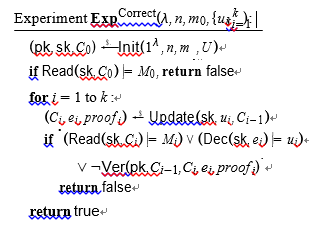
我们还定义了两个确定性函数Read（sk，C）→M和Dec（sk，e）→u，它们从C读取每个值作为明文存储器M并解密更新密文，  
分别。 我们仅在我们的正确性和安全性定义中使用这些操作。 它们不是核心PVORM接口的一部分。

我们定义一个PVORM关于有效更新序列的正确性。 如果当M0 =（m0，...，m0）和Mi = f（ui，Mi-1）时，更新序列{u0 k对于m0是有效的，则对于所有0≤i≤k定义Mi。

一个PVORM是正确的，如果所有初始值m0和所有更新序列({ui k})对m0有效，

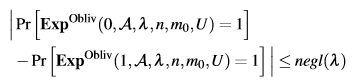
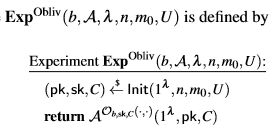
Pr[**Exp**Correct(*λ, n, m*0*,* {*ui k* )] = 1

其中**Exp**Correct(*λ, n, m*0*,* {*ui*}*i*=1) 被定义为试验



与{M0，...，Mk}定义如上。 换句话说，如果更新正确地转换f所定义的f，那么PVORM是正确的，Ver将验证这些更新。

**不确定性。**Solidus需要一个能够在新的环境中实现ORAM保证的结构，即使是适应性的对手也是如此。直观上，我们要求PVORM保证任何两个自适应选择的有效更新序列都不会导致难以区分的输出。在形式上，我们说如果对于所有的PPT对手A来说PVORM是没有意义的，那么对于所有的n∈N，m0和U，都有一个可以忽略的negl（λ）

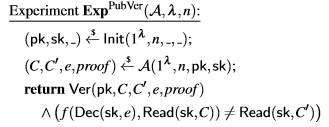
其中 

其中Ob，sk，C（·，·）是初始状态为S←C的有状态预言者。输入（u0，u1），Ob，sk，C执行（Cr，e，证明）← - 更新 ，S），更新S←Cr，并返回（Cr，e，证明）。 如果任何Cr未定义，则实验中止。

这个定义是ORAM文献中提出的适应性版本。

公共可验证性。我们的安全定义的最后一部分是公开的可验证性。 直观地说，我们要求每个更新都能证明所执行的更新是有效的并且是所声称的更新。 对于所有PPT对手来说，PVORM正式可以公开验证： Pr[ExpPubVer(A,λ,n)]≤negl(λ)

其中ExpPubVer(A,λ,n)被定义为：



这对应于正确执行更新的ZK证明的可靠性。

**4.2 Solidus实例化**

在Solidus中，我们通过将Circuit ORAM的结构与几个GSP结合起来而来实例化一个PVORM。循环ORAM将数据块放入组成二叉树的桶中。 它通过交换树中的路径来执行更新（驱逐）。 这种结构导致了良好的性能，其原因有两个：更新操作是对数的计数，公钥密文的两两交换提供了高效的ZK证明正确性。 图2显示了Solidus的PVORM是如何构建和更新的。

每个数据块都拥有一个账户的唯一标识符和余额。随着区块被洗牌，这一对值必须一起移动，所以Solidus采用可验证的交换算法来增加交换有序密文对（见附录A.4）的El Gamal密文。

Solidus限制每个更新以修改一个账户余额，并要求余额保持在固定的范围[0，N]。 为了使更新可以公开验证，银行首先将所需账户移动到一个确定性的固定块，通过交换每个块沿着电路ORAM访问路径的位置。 在数据处于确定位置时，银行更新账户余额，并对所得密文生成一组包含证明，证明其在合法范围内（见附录A.5）。 最后，银行执行电路ORAM的驱逐算法重新插入更新的账户。 这又需要用一组树路径来交换固定块。在附录B中，我们详细介绍了这种结构，并证明它是正确的，不经意的，可公开验证的。

存储溢出。电路ORAM假定有限大小的存储器，但是如果存储器溢出则可能丢失数据。 这导致了正确的概率定义; 正确的行为只发生在数据不丢失的情况下。 由于数据丢失的概率在存储容量上可以忽略不计，因此定义对于设置是合理的。

Solidus也采用有限的藏匿。 存款必须放在分类帐上，所以更改其大小会泄漏有关账户位置的信息，从而泄漏交易图表。 然而，在Solidus中，无论多么罕见，数据丢失都是灾难性的，所以我们采取不同的方法来存储溢出。 当存储溢出时，而不是丢失数据，我们在树中插入一个更深的账户。 这种插入是公开的，所以它泄露了常规驱逐不足以及单个真实账户的位置（尽管不是该账户的身份）。

Solidus继承了隐藏ORAM的隐藏溢出概率，隐藏大小可以忽略不计。 正如我们在第7节中所展示的那样，PVORM更新的性能在存储大小上是线性的，这给Solidus带来了直接的性能 - 隐私权衡。 而且，只有适度的储藏大小使得溢出变得极不可能。 使用大小为3的桶时，大小为25的容器将溢流概率降低至大约2−64。

1. Solidus 协议

我们现在介绍Solidus协议。这个描述在很大程度上依赖于密码学原语，其细节我们遵循附录A.我们做这个选择既简化了解释，又用几个实例离开操作，比如范围化的证明——抽象概念。

**银行的状态。**银行Bi的状态由加密密钥对（ePKi，eSKi），签名密钥对（sPKi，sSKi）和一组账户组成。每个账户都有一个唯一的账户标识符和一个余额。 为了简单，我们使用Uj的公钥pkj作为它的标识符。

每家银行都维护自己的PVORM，包含每个账户的信息。 4.2节描述了PVORM结构。

**交易请求。**由于Solidus是银行中介的，我们在Bs必须向Bs发送一个请求，以便将$ v发送给Br的Ur。 该请求包括：

•一个专门的ID txid

•将Enc（ePK，$ v），$ v在B的公共密钥下加密

•Enc（ePKr，pkr），是Br的公钥下Ur的ID的密文

•A hidden-public-key signature signed with sk*s* (see Appendix A.3).

收到请求后，Bs必须验证请求——检查txid是全局唯一的，并且0 ≤ $*v* ≤ $*bs* - 并启动交易处理结算过程。

**交易结算。**图3显示了交易结算的结构。 Bs生成$ v≥0的证明，在ePKr下重新加密$ v，并向Br发送（txid，Enc（ePKr，$ v），Enc（ePKr，pkr））。 然后两家银行（同时）更新各自的PVORM，并尽心他们各自的更新，并将所有相关的证明和签名张贴到分类帐上。 一旦全部交易被分类账所接受，资产已经转移，交易已经结算。

交易IDs。为防止重放攻击，Solidus要求每个事务都有一个全局唯一的交易ID。 如果这个ID只是一个随机的比特串，这将要求每个银行在系统的整个生命周期中检查每个交易的ID与之前每个交易的ID。 为了避免这种成本增加，Solidus使用两部分交易ID：时间戳和随机数。然后交易只在TΔ的特定窗口内有效。对于具有txid =（T，id）的交易，如果交易在*T*now时刻被处理，则只有在*T*now -TΔ<T < *T*now时才有效。 时间戳必须是过去的，但不能超过TΔ。 这使得验证银行在搜索ID时只能回看TΔ以防止重放攻击。

开放和关闭的账户。银行不断开设新账户，所以Solidus必须支持这一点。要创建一个账户，银行必须将该账户插入其PVORM。 我们的建设使这个简单。 Bi公布新的ID，ID为可验证的加密串，余额为0.然后，通过替换虚拟值，将这个密文对插入到PVORM中。 关闭账户Bi只是公开验证账户的身份并用虚拟值代替PVORM。

**5.1安全定义**

我们用图4所示的理想功能FSol来定义Solidus的安全性。我们在具有理想分类账FLedger的混合世界中定义协议本身（参见附录C）。 对于对手A和环境Z，我们让Hybrid*A,Z* (*λ* )表示A和Ideal*S,Z* (*λ* )的混合体系转录本，是由在理想世界中运行的模拟器S产生的转录本。

定义1. 我们说Solidus安全地模拟F*Sol*，如果对于所有现实世界的PPT攻击者A和所有环境Z，存在一个模拟器S，使得对于所有PPT识别器D，



我们假设Solidus只使用通用组合（UC）NIZK。 先前的工作表明，通过使用Fiat-Shamir启发式，可以将GSPs转换成UC-NIZKs，并在共同初始化器提供的公共密钥下包含密文的密文。 由于Solidus已经采用了这种可信的初始化，并且包含大多数操作的密文，所以确保UC-NIZKs的性能影响是最小的。 这使我们能够证明通用可组合性（UC）框架中的安全性。

定理1. Solidus协议满足定义1假定ROM中的DDH-hard组。

我们提供了附录C中定理1的一个证明。

**6个性化**

我们现在进行一些优化以使Solidus更实用。 其中一些优化只适用于某些用例，但在适用的情况下可能会导致显着的加速。 我们在第7部分的评估中包含了更简单的优化。

**6.1预先计算随机因素**

Solidus的一个很大的计算成本就是在更新PVORM时重新对密文进行随机化。 幸运的是，El Gamal的同态性质允许我们通过将密文与新的组身份加密相结合来重新随机化密文。 也就是说，在给定公钥/私钥对（pk = gsk，sk）和密文c =（α，β）的大小为p的群G =（g）中，我们可以生成一个重新随机化的cr 通过随机选择r←Zp并令cr =（α·pkr，β·gr）。

方便的是，计算（pkr，gr）只需要知道组G，生成器g和公钥pk，在系统的整个生命周期中，所有这些对于给定的银行是静态的。 这意味着我们可以预先计算  
这些单位密文通过乘以预先计算的值重新随机化。

由于系统在其生存期内承认了无限数量的事务, 我们必须不断地生成这些随机因素。许多金融系统都有非常可预测的高和低负载时间 (例如, 夜间的通信量通常很轻), 因此他们可以利用其他闲置的硬件在低流量时计算随机因素。如果算进程可以生成比应用程序在一个适中的时间框架 (例如一天) 消耗的更多的随机化对, 我们可以大大提高性能。

**6.3 事务流水线**

在一个设置中, 任何形式的故障都是罕见的 (即使拜占庭式的失败是可能的), 我们可以进一步优化。同样, 对于大型商业银行来说, 这种设置是非常合理的。

Solidus要求对单个银行的事务进行顺序处理, 因为 PVORM 更新必须是顺序的, 才能生成有效的校样。给定事务T1 , 后跟t2, 以便 B 处理t2它需要在t1之后的 PVORM 状态。然而, 它并不需要相关的证据。因此, 如果 B 假定t1将结算, 则在为t1生成校样时, 它可以在早期开始处理t2 。当故障是罕见的, 这是一个合理的假设。虽然此技术不会减少事务延迟, 但它可以大幅提高吞吐量。此外, 确定更新的 PVORM 状态需要主要重新随机化密文, 使此优化在与计算 (6.1 节) 结合时特别有效。

当发生故障时, 它会影响性能而不是正确性。如果 T1因某种原因而中止, 则 t2将在第一次结算后才会被终止。这意味着 B 可以立即识别问题并重新处理t2-以及以下任何事务且没有t1。这种后处理可能会导致重大但暂时性的性能降级, 因此只有在出现故障时才适用此优化。

**7实验**

我们现在介绍我们的PVORM和Solidus实现的性能结果。 对于我们在c4.8xlarge Amazon EC2实例上运行的所有实验，采用6.1节中讨论的预计算优化。 这些基准不包括计算新的加密随机因素的时间。

**7.1 PVORM 性能**

在不同的参数和并行度下, 我们测量了 PVORM 更新和运行的具体性能。

**桶和藏匿的大小。**图5显示了我们的 PVORM 的单线程性能, 因为我们改变了桶和藏匿的大小。如预期的那样, 较大的存储桶速度较慢, 且运行时与隐藏大小呈线性增长。由于桶和藏匿的大小决定了藏匿溢出的可能性, 这提供了一个性能与隐私的权衡。

**树深度。**图6显示了我们的 PVORM 作为容量刻度的单线程性能。正如所料, 二叉树结构会产生明显的对数缩放。

**并行。**我们的 PVORM 建筑适合于高度平行的操作。单个更新包含大量可以独立创建或验证的 NIZKs。图7显示了一个具有不同工作线程数的单个 PVORM 的性能。在每个测试中, 只有一个协调线程, 很少工作。

因为每个成对交换的证明都可以独立计算或验证，所以我们预计性能可以超过10个线程 - 可能高达100个。我们在两个原因的组合下停止10。首先，PVORM操作是CPU限制的，因此在CPU内核数量之外添加线程不会产生有意义的加速。其次，我们的原型实现不会分配给多个主机，而是不能很好地扩展到多CPU体系结构。由于c4.8xlarge的EC2实例有两个10核心的CPU，所以我们只提供了10个工作线程的扩展。请注意，在10个工作线程中，总共有11个线程，所以有些工作可能无法在同一个CPU上有效地并行化。 这可能解释了这种情况下的一些缩小比例。

**证明大小和内存使用情况。**对于大小为3桶、大小为25的存储和容量为215的 PVORM, 带有校样的单个 PVORM 更新是 196 kb (或 117 kb, 如果压缩)。为了生成更新, 我们的原型需要一个完整的 PVORM 在内存中的副本。尽管如此, 内存消耗峰值只有 880 MB。

**7.2 Solidus系统性能**

现在, 我们向2到12家银行展示了全分布式Solidus系统测试的性能。每个银行都在其自己的 c4.8xlarge EC2 实例上运行, 并维护一个 PVORM 大小为3的存储桶, 大小为 25, 容量为215。这些参数允许进行合理的测试, 并使隐藏溢出概率大约为2−64。我们通过让每个银行的主机也运行ZooKeeper节点来维护分类帐。我们没有试图调整ZooKeeper或优化分类账本沟通.

我们测试此配置, 每个银行都满载着传入和传出事务。如6.2 节所述, 在离线状态下执行部分或全部事务验证通常是合理的, 因此我们还会关闭联机验证来测试性能。

图8包含了这些测试的结果。通过定期的在线验证, 性能会得到改善, 直到所有cpu都被饱和验证第三方事务处理后, 之后点扩展速度会减慢。使用离线验证, 交易结算速度更快, 其他银行对现有银行施加较低的开销, 从而提高了规模。

通过对各银行进行跨多个机器的分布式交易验证, 提高容量和提高吞吐量, 可以进一步改善这些结果。流水线的事务 (如6.3 节所述), 如果银行还在多个主机间分布证据生成, 则可以大大提高吞吐量。(这种分配不太可能在没有流水线的情况下提供任何好处。我们没有对这些选项进行基准测试, 因为我们的原型不支持将单个银行分发到多个主机。

**7.3 zk-SNARK比较**

我们最后比较了我们的原型的性能, PVORM 实现与zk-SNARKs。这种方法得益于简洁的校样和简短的验证时间, 但证明生成成本很高。

简单地把我们的电路ORAM PVORM 建设和转换所有的证据, 以 zk-SNARKs将需要不必要的和昂贵的加密操作。zk-SNARKs能证明一个任意电路的正确应用, 我们使用一个紧凑 Merkle 树结构, 我们在附录 D. 1 中详细描述。

我们使用等效的安全级别实现了这一建设, 我们的 GSP-based PVORM。表1显示了在 c4.8xlarge EC2 实例上运行的性能. 我们看到, 虽然验证速度非常快, 但即使是高度并行的证明生成也比 GSP PVORM 慢200倍。为了提高整个系统的吞吐量, 系统需要验证每一个证据200次左右。在我们预期的用例中, 多数银行将保持总账, 因此这一速度明显放缓。此外, 额外的硬件可以允许银行并行地验证大量的 GSP 交易, 但对zk-SNARKs没有什么好处。

**8相关工作**

我们将我们的工作与其他几种匿名数字货币进行比较。 正如我们在第2.3节和第4节中讨论的ORAM，我们在这里不再这样做。

**匿名电子现金。**匿名电子现金最初是由Chaum提出的，后来被其他人改进。已知的电子现金方案依赖中央化的信任，这意味着每个交易都必须涉及一个单一的权力机构。在一些方案中，受托人可以使用私钥撤销用户或交易的匿​​名;这个功能可以追踪非法活动。我们在Solidus中的设计目标是不同的。 Solidus用户可以选择他们信任的银行，而不必信任中央银行，而是保留系统中所有其他实体的匿名性。Solidus用户只有信任他们各自的银行才能获得隐私和可用性。虽然恶意银行可以阻止用户进行交易，但用户对数字签名的要求可以防止银行滥用用户资源。另外，如上所述，用户可以在审计师的帮助下证明资产拥有权，并且如果资金被不适当地冻结，则将其账户转移到另一家银行。

**匿名分散密码加密货币。**Zcash及其先前的Zerocash提供了一个匿名的分散加密货币。 具体而言，它依靠预处理zk-SNARKs来确保资金节约，防止双重支出，并隐藏交易价值和交易图。 在霍克，科斯巴等人。指出Zerocash的安全定义和证明存在缺陷，并构建自己的类似的非集中匿名加密货币（保护隐私的智能合约）。

Zcash和Hawk都依靠预处理zk-SNARKs，因此系统需要可靠的设置。 而本-萨松等人。 指出可信任安装可以通过多方计算分散，这是一个复杂的过程，尚未在任何重要的系统中执行。而且，正如我们在第7.3节中对Solidus的zk-SNARK变体的研究中所展示的那样，zk-SNARKs比Solidus中使用的GSP更加昂贵（两个数量级）。此外，Zcash和Hawk不打算像Solidus那样提供可审计性。 按照设计，它们不记录资产上的链条，只记录承诺。

除Zcash和Hawk之外，其他计划提供各种形式的分散化密码或非密码混合形式，以增强匿名性并帮助模糊交易图。但是这些系统并不能提供完全的安全性，而且通过统计分析可能会破坏匿名性。目前尚不清楚需要多少混合来抵制统计分析，并达到实践中所期望的匿名水平。相比之下，Solidus获得了更强的匿名性保证。来源银行和目的银行都被披露，但没有其他交易信息被泄露。

**机密交易。**一类称为“机密交易”的方案隐藏了交易金额，而不是提供交易图的隐私。 （这些方案的一个子集，使用非正统的加密工具，没有附带的证据，已经被证明是有缺陷的，并且一直在进行“构建，中断，修复”的循环 - 例如，化身被破坏了两次。 方案隐藏交易金额; 这个方案是相似的，但是更直接的使用和继承了GSP的可证明的安全属性。

**9结论**

我们推出了Solidus，这个系统解决了广泛使用区块链交易系统的一个主要障碍，它们严重缺乏交易图的机密性。与以前的方法（例如Zcash）不同，Solidus专门针对现代金融交易和结算系统的结构和性能要求。 Solidus的关键创新是可公开验证的不记名RAM（PVORM），这是ORAM的推广。 PVORM支持可公开验证的计算内存外包，为区块链交易系统设计提供了全新的方法。 Solidus使用数据结构大小与账户数量呈线性关系的PVORM（而不是系统中的交易数量），如Zcash，证明计算时间比zk-SNARK快两个数量级。我们将Solidus的安全性定义为理想的功能，并在UC框架中证明其安全性。最后，我们介绍了在分布式总账（ZooKeeper）上运行完整Solidus协议的一系列优化和实验，展示了Solidus扩展到实际工作负载所需吞吐量的能力。我们相信Solidus是建立强有力的可验证和完全可审计的银行中介账簿交易系统的第一个可行的方法。

**致谢**

这项工作是由国家科学基金会拨款CNS-1314857，CNS-1330599，CNS-1453634，CNS-1518765，CNS-1514261，CNS-1514163和CNS-1564102，ARO拨款W911NF-16-1-0145，一个帕卡德 奖学金，Sloan奖学金，Google教师研究奖，VMWare研究奖，以及Chain，IBM和Intel的IC3赞助。 我们要感谢TradeWind Markets的Matthew Trudeau和Abishek Kumarasubramanian以及Digi- asset Asset Holdings的Shaul Kfir和Tama的Blummer耐心地解释了金融行业的需求，背景和要求。 我们还要感谢Eleanor Birrell，Philip Da-ian，Joshua Gancher，Andrew Morgan和Isaac Sheff的精辟评论和帮助编辑。

**参考**

[1] Personal communication with Matthew Trudeau, Presi- dent, TradeWind Markets.

[2] Personal communication with Shaul Kfir, CTO, Digital Asset Holdings.

[3] Personal communication with Tama´s Blummer, Chief Ledger Architect, Digital Asset Holdings.

[4] I. Anati, S. Gueron, and S. Johnson. Innovative technol- ogy for CPU based attestation and sealing. In *Proceed- ings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, 2013.

[5] B. Barak, R. Canetti, J. B. Nielsen, and R. Pass. Uni- versally Composable Protocols with Relaxed Set-up As- sumptions. In *FOCS*, 2004.

[6] S. Barber, X. Boyen, E. Shi, , and E. Uzun. Bitter to Better—How to Make Bitcoin a Better Currency. In *Fi- nancial Cryptography*, 2012.

[7] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green,

I. Miers, E. Tromer, and M. Virza. Zerocash: Decentral- ized anonymous payments from Bitcoin. In *IEEE Sympo- sium on Security and Privacy*, 2014.

[8] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and

M. Virza. Snarks for C: verifying program executions succinctly and in zero knowledge. In *CRYPTO*, 2013.

[9] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and

M. Virza. Secure Sampling of Public Parameters for Suc- cinct Zero Knowledge Proofs. In *IEEE Symposium on Security and Privacy*, 2015.

[10] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Suc- cinct Non-Interactive Zero Knowledge for a von Neu- mann Architecture. In *USENIX Security*, 2014.

[11] E.-O. Blass, T. Mayberry, G. Noubir, and K. Onarli- oglu. Toward Robust Hidden Volumes Using Write-Only Oblivious RAM. In *CCS*, 2014.

[12] F. Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT*, 2000.

[13] J. Camenisch, S. Hohenberger, and A. Lysyanskaya.

Compact e-cash. In *EUROCRYPT*, 2005.

[14] J. Camenisch, A. Kiayias, and M. Yung. On the Porta- bility of Generalized Schnorr Proofs. In *EUROCRYPT*, 2009.

[15] J. Camenisch, A. Lysyanskaya, and M. Meyerovich. En- dorsed E-Cash. In *IEEE Symposium on Security and Pri- vacy*, 2007.

[16] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *CRYPTO*. 1997.

[17] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.

[18] M. Castro and B. Liskov. Practical Byzantine Fault Tol- erance. In *OSDI*, 1999.

[19] D. Chaum. Blind signatures for untraceable payments. In

*CRYPTO*, 1982.

[20] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *CRYPTO*, 1990.

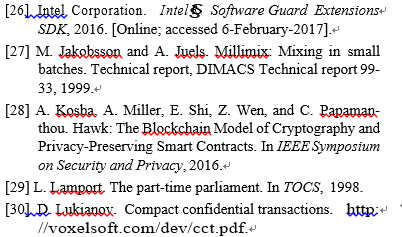
[21] I. Damga˚rd. On *σ* -protocols. *Lecture Notes, University of Aarhus, Department for Computer Science*, 2002.

[22] M. del Castillo. Overstock just closed its first day of blockchain stock trading. *Coindesk*, 16 December 2016.

[23] A. Fiat and A. Shamir. How to prove yourself: Practi- cal solutions to identification and signature problems. In *EUROCRYPT*, 1986.

[24] G. Hinterwa¨lder, C. T. Zenger, F. Baldimtsi, A. Lysyan- skaya, C. Paar, and W. P. Burleson. Efficient E-Cash in Practice: NFC-Based Payments for Public Transportation Systems. In *PETS*, 2013.

[25] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In *USENIX ATC*, 2010.



[31] G. Maxwell. Confidential transactions. [https:](https://people.xiph.org/%7Egreg/confidential_values.txt)

[//people.xiph.org/~greg/confidential\_](https://people.xiph.org/%7Egreg/confidential_values.txt) [values.txt](https://people.xiph.org/%7Egreg/confidential_values.txt).

[32] G. Maxwell. CoinJoin: Bitcoin privacy for the real world.

bitcointalk.org, August 2013.

[33] G. Maxwell and A. Poelstra. Borromean ring signatures. [https://github.com/Blockstream/](https://github.com/Blockstream/borromean_paper) [borromean\_paper](https://github.com/Blockstream/borromean_paper).

[34] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas,

H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. Inno- vative instructions and software model for isolated execu- tion. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, 2013.

[35] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko,

[36] D. Mulligan. Know your customer regulations and the international banking system: Towards a general self- regulatory regime. *Fordham Int’l LJ*, 22:2324, 1998.

[37] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. <http://bitcoin.org/bitcoin.pdf>, 2009.

[38] D. Ongaro and J. Ousterhout. In Search of an Understand- able Consensus Algorithm. In *USENIX ATC*, 2014.

[39] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, 1991.

[40] V. Phegade and J. Del Cuvillo. Using innovative instruc- tions to create trustworthy software solutions. In *Proceed*

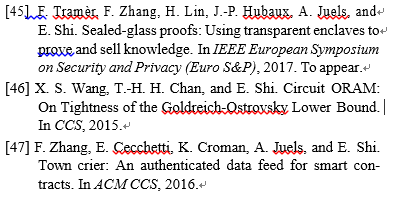
[41] T. Ruffing, P. Moreno-Sanchez, and A. Kate. CoinShuf- fle: Practical decentralized coin mixing for Bitcoin. In *ESORICS*, 2014.

[42] C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.

[43] E. Shi, T.-H. H. Chan, E. Stefanov, and M. Li. Obliv- ious RAM with *O*((log *N*)3) Worst-Case Cost. In *ASI- ACRYPT*, 2011.

[44] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren,

X. Yu, and S. Devadas. Path ORAM – an extremely sim- ple oblivious ram protocol. In *CCS*, 2013.

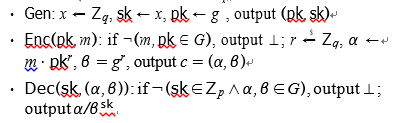


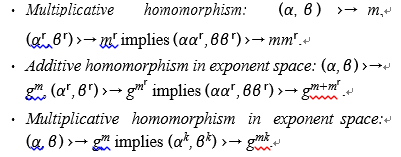
**一个加密基元**

我们现在描述Solidus中使用的基本密码学原语。 这些原语在由（线性）安全参数λ确定的阶p的乘法循环群G =（g）上操作。 正如我们所解释的那样，我们的构建块要求G.对于G.的决定性Diffie-Hellman假设（为了防止使用Pohlig-Hellman算法的子组攻击，p通常是主要的）。在我们的Solidus实现中，G 是secp256k1椭圆曲线组。

**A1 El Gamal加密和账户平衡表示**

El Gamal密码系统（Gen，Enc，Dec）如下：



如果决定性Diffie-Hellman（DDH）问题对于G是困难的，那么El Gamal加密在语义上是安全的。 埃尔加马尔密文是可塑的，但是，在我们的建设一个有用的功能。 具体而言，El Gamal有几个有用的同态。 令（α，β）>→m表示对于r∈Zp，（α，β）解密为m，即（α，β）=（m·pkr，gr）。 接下来是：

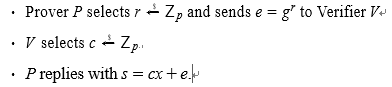
观察到，在不知道sk的情况下，密文（α，β）>→m的重新加密可以使用多重同态来实现：令r← - Zp，计算新的密文（αr，βr）=（pkr，gr ）>→1，然后令（A，B）=（ααr，ββr）。 观察（A，B）>→（m×1）= m。

**账户余额表示。**Solidus中的加密基元依赖于指数空间中的账户平衡表示, 以便在上面所示的指数空间中利用加法同态。因而账户平衡 $v 被编码在形式g$ , 并以 El 的密文为代表 (g $v pkrgr,r ∈) 因此, 对账户余额进行解密需要解决g$ v 上的离散日志问题。虽然一般这是很难在G, 如果 $v 是已知的相对较小的 (例如, 0 ≤ $v < 230), 然后余额可以使用可管理大小的查找表对其进行解密.

**A2 广义身份证明（gsp）**

广义身份证明是Σ-协议的一种类型，即3移动诚实验证者零知识（HVZK）证明（通常更具体地定义为具有特殊健全性的特殊3移动HVZK证明）。 GSP专门针对分组日志问题和变体难以处理的组织进行操作。 我们注意到，在这里我们只考虑GSPs在一个循环的素数秩序组，避免关于复合秩序组的警告。

给定x← - Zp和y←gx，存在一个简单的Σ协议来证明只有知道y = gx的验证者的x的知识：

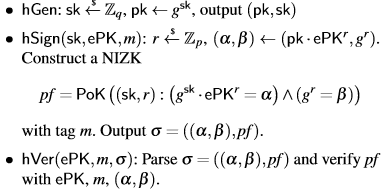


验证V然后检查gs = eyc。 这个协议是用GSP的语言规定的，使用的符号是：PoK(x : y = gx) ,

有可能构建高效的GSP这些谓词连接和分离，并且其他谓词的有效构造也被显示出来。 另外，可以使用Fiat-Shamir启发式将随机Oracle模型（ROM）中的GSP转换为NIZK，其涉及在Σ-协议的第一步中对证明者的消息进行散列。 也可以附加一个我们称之为标签的附加值给消息进行散列。 例如，具有标签m的NkK版本的PoK（x：y = gx）是m上的身份签名。 在Solidus中，所有的ZPK都是这样的NIZK，我们在附录的剩余部分中隐含了这个事实。

**A3 隐藏的公钥签名**

为了在不泄露发送用户的情况下对事务进行身份验证, Solidus使用了隐藏-公钥-密钥(HPK) 签名方案。这个简单的方案允许签名者对在银行的公钥 ePK 下加密的签名公钥 pk 进行签名, 即密文 (α, $ Enc(ePK,pk)。带有公钥 ePK的 HPK,签名方案(hGenhSign,hVer)

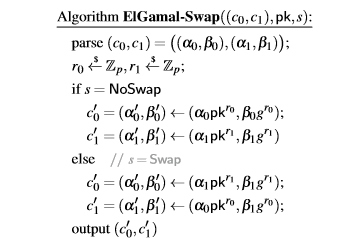


这种形式的 HPK 本身并不十分有用, 因为接收方只知道对某些密钥生成了有效的签名, 但对密钥一无所知。

事实(α) ,β是 pk 下的 ePK 密码, 但是, 这使得这种签名有两种用处。首先, 当 U 请求事务时, 它允许 B 解密 pk 并识别 u。其次, 它允许 B在(α,β) 上生成明文等效性证明,以及与平衡 B 在其 PVORM 中更新。第二个属性验证其余额是否已更新的用户知道 sk, 从而使其成为有效的签名.

**A4 El Gamal互换**

在Solidus中进行证明生成和验证所需的大部分计算都是专门用于我们所谓的El Gamal交换。 操作ElGamal-Swap将有序对的El Gamal密文（c0，c1）=（α0，β0），（α1，β1），相应的公钥pk和值s∈{Swap，NoSwap}。 它输出一个新的有序对，在pk下重新加密，并带有相同的底层明文。 如果s = NoSwap，则明文顺序与原始密文相同，否则交换。 算法如下：



通过指定的GSP，可以证明输入/输出对（c0，c1）和（c0 0，c0 1）的El Gamal Swap的正确执行是有可能的。

在Solidus中，由于账户由账户的公钥和账户余额中的一对密文表示，事实上我们需要执行双重El Gamal交换，这意味着两对密文交换使用 s的相同值。 正确性的证明涉及到GSP的单一交换的简单扩展。

双交换证明需要13个椭圆曲线乘法，而验证需要18个。

**A5 范围证明**

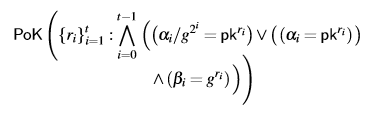
有许多协议 (例如 [12]) 用于验证表单的声明 (x = y gx ∧ l0 lp )。

在Solidus中, 在概念上简单的机密事务处理方法 [31] 中, 我们使用 GSP 来证明一个 El 的密文c =(α,)=()/", ) , 它表示帐户余额 0 ≤ $v。具体来说, 为了排除模块化的环绕, 我们证明了整数 $v ∈ [0,2t) , 它确定帐户余额的上限。在我们的原型中, 我们设置t = 30.

我们现在描述我们使用的 GSP 以按位方式完成此范围证明。首先, 我们观察到显示为密文的 () (αi,我, βi→7 ∈ , /下/0/ 2i} 在公钥 pk 下, 足以证明:



因此GSP有



证明了

这样$*v* ) ∈ [0*,*2*t*, 即(*α,β*) 是账户余额的

密文$*v* ∈ [0*,*2*t* ).

此范围证明需要 5+10t椭圆曲线乘法和t加密 (除非使用预计算否则需要2乘法), 而验证需要 7+12t乘法.

我们表示这样的证明: 密文c对值进行加密[0,2t) (在指数空间中) 由 RangePf(c,t ).

**A6 ORAM电路**

Solidus的主要数据结构用于存储账户余额的分类帐是一个 PVORM 基于结构的电路ORAM [46]。然而, PVORM 的目的是提供非常不同的担保比经典ORAM。ORAM使具有有限本地内存的客户端能够在数据结构C中将一个大的虚拟内存M项外包给一个更强大的外部设备, 通用称为服务器。目标是使客户端能够以尽可能少的本地存储来存储M机密.

ORAM确保访问模式机密性; 尽管它能够观察客户端对C的访问, 但服务器不了解关于客户端访问M中的块的模式的任何信息 (无不可)。使用对称密钥密码对C中的块进行加密, 以确保数据的保密性。但是请注意, 仅加密并不隐藏访问模式。m被构造为一组块 m[[ [1],m2,,...,m N]。例如, C[idx] 仅加密当前值M[idx], 例如, 服务器就会知道每次客户端读取或写入M[idx], 因为它将看到客户端访问C[idx].

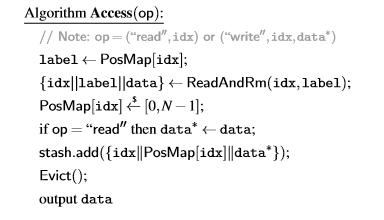
因此, 为了实现访问模式的保密性, ORAM的实现需要更复杂的方法。

在此方法中, C表示为深度树L = 日志N +1 (N被假定为2的幂)。树中的每个节点都包含一个桶, 它具有用于存储块的b插槽, 其中b是系统参数。这些插槽中的大多数在给定时间是空的, 这是一个重要的事实, 我们将在下面看到.

块采用窗体 idxk标签k数据, 其中 idx 是块的索引-对应于其虚拟内存插槽的值 idx M [idx]中, 标签标识树中的一个叶, 该路径位于该块所在的根目录中C中, 数据存储块内容.

客户端维护一小部分本地内存, 称为存储库, 它是处理从C溢出的缓冲区。客户端还存储一个位置映射PosMap, 一个数据结构, 如 PosMap[idx] = 标签。即, PosMap 将给定块的索引 idx 映射到M中相应的叶值标签。(PosMap 可以以递归方式存储在服务器上的单独ORAM中, 以减少存储开销, 这一功能与 PVORM 无关.

读取和写入涉及客户端在C上的相同基本操作访问, 如下所示。



在这里, ReadAndRm 读取包含目标块的C中的完整路径, 并在隐藏时删除块 (沿路径加密块). add 执行在隐藏中添加块的明显操作。可以任意或地执行退出。随机方法选取两个叶leafl 和leafr , 它们分别从三的左右半部分中随机抽取, 并在它们定义的 root-to-leaf 路径中执行所谓的逐出传递。确定性方法 (我们在我们的 PVORM 构造中采用) 是相同的, 但它在旋转的确定性顺序中选择leafl 和leafr , 该顺序旨在将逐出通道设置为尽可能远离彼此的连续通道 仍然通过足够的访问旋转每一片叶子。

在给定路径上的逐出传递涉及在相邻路径元素的对上执行交换, 从树的顶部到底部, 并将其作为特殊的 "级别 0", 即, 位于根之上。这些互换的目标是将块沿着路径移到尽可能低的水平。块被 "拾起" 并且通过连续的互换移动到最低的点, 因此它保持在由标签定义的路径上, 并且有一个空的槽可供它使用。此时它被 "丢弃"-插入到该级别的桶中。一个区块可以从插槽中提取, 最后一个是下降或交换可能继续, 直到另一个区块, 可以进一步推下路径。在两条路径上而不是一条路上执行逐出的原因是确保在C中的块在全局范围内保持足够深, 以防止大量溢出到隐藏中。

在ORAM电路中的这个处理步骤实际上是相当复杂的。客户端没有有关块在C中驻留位置的完整本地信息, 因此必须使用元数据计划交换。(这种并发症不会出现在 PVORM, 但是, 正如我们下面解释。

其他树 ORAMs, 如路径ORAM, 主要不同于他们使用替代驱逐战略。在ORAM电路中使用掉期的做法特别有利于Solidus的高效 NIZK 生产, 但这也是Solidus PVORM 使用的原因。

**B Solidus PVORM建设**

我们现在介绍了Solidus中使用的 PVORM 结构的详细信息, 并证明它是一个正确的、易失的、可公开验证的 PVORM。我们注意到, 可以从任何ORAM、ZK 证明系统和加密方案 (对称或公钥) 中构建 PVORM。我们的 PVORM 在Solidus, 但是, 是为了确保高效的证明计算支持高出入量。为此, 我们使用ORAM电路, 非交互式广义身份证明, 和 El 的加密。

从上面回想一下, ORAM电路由一个桶的二叉树组成, 每个都包含一个固定数量的数据块。每个位置都包含数据块或虚拟值的加密。每个逻辑数据块都与树中的单个叶相关联, 并在该叶的路径上物理地驻留在某处。为了访问逻辑数据块 (读取或写入), 客户端将沿着路径到关联的叶读取所有块。然后, 客户端将访问的逻辑块与一个新的随机叶关联起来, 并写出沿着被访问路径和树中的其他两个确定性路径的所有块的新加密。在这些写入过程中, 客户端将现有的数据块把到叶子上, 同时保持每个实际数据块在其相关叶的路径上保持不变。这些搬迁可以在树的深度上进行大量的物理内存位置的两两交换。我们利用在我们的 PVORM 建设中通过成对互换进行的驱逐的能力。

**B1 结构**

在Solidus中, 每个银行都保留自己的 PVORM 来存储用户账户余额。由于 PVORM 与单个银行具有唯一的关联, 因此我们为4节中指定的密钥对使用简单的 El。每个逻辑地址都由账户 ID 指定, 每个数据块本身就是一个账户余额。为了存储这些, 每个数据块包含一对加密: 一个账户 ID 和一个余额。我们将最大余额限制为相对较小的值 (如 230或 240)。这允许我们对指数空间中的平衡进行加密, 创建一个加法同态, 同时仍然允许解密 (使用查找表)。让t表示最大余额的二进制日志。

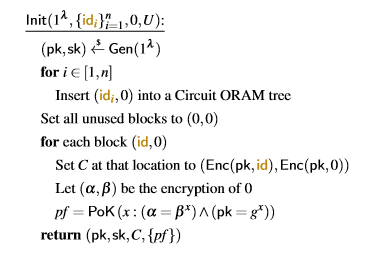
因此, 我们将M作为从账户 id 到账户余额的映射。我们定义了 PVORM 更新函数f((id,) ,, . [ ] 在m和 (中, id与mid]+[ id M [id]+$v ) ) ∈ [0,2t。否则) ,,((未定义 f (id 直观地, f更新单个

在有效范围内的任何值的账户余额。

如4节所述, 我们使用的是一个固定大小的公共藏, 而不是由ORAM电路所假设的动态私有的。为了简单起见, 我们将此隐藏合并到树的根节点中。储存的每个数据区块的形式与树中的相同。另外, 我们还使用了一个可识别的固定块。此块只是每个路径上存在的单个确定性块。它可能是根桶/藏的一部分, 也可能是它自己的单独位置。

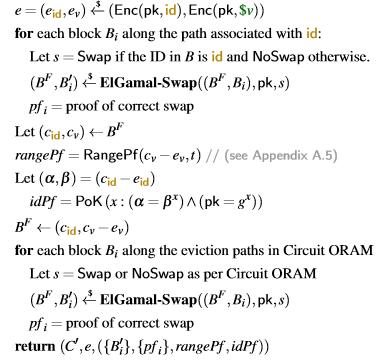
我们现在描述4节中定义的每个操作的实现。让 (Gen,,Enc12月) 是标准的 El.

**结构 1(Solidus PVORM).** 我们总是将所有余额初始化为0。更新空间U由帐户 ID/事务值对组成, 值介于最大余额和负数之间。初始化收益如下:



让M = 读取(sk,C)。我们注意到, 更新(sk,) ,u,只有在f(u ,M ) 已定义。给定u、sk 和C, 此属性很容易检查, 因此我们的定义只是假定在输入上定义了更新, 而不显式检查。让BF 成为可分辨的固定块。为了简单起见, 假设与 sk 相关的 pk 是可用的 (或者是作为 sk 或可的一部分存储).

更新值(sk,u,C):



只需验证更新输出中包含的所有 NIZKs, 并验证在两组交换之间正确计算更新的BF , 即可执行验证。

**B 2 安全证明**

我们现在证明了前一节所给的结构的安全性。

定理 2 (PVORM 正确性)。构造1是正确的PVORM.

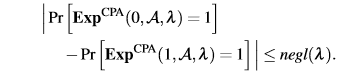
证明. 以下属性可确保正确性.

· ORAM电路是正确的, 当储存不溢出和建设1修改ORAM电路泄漏事务图形信息, 而不是丢失数据溢出.

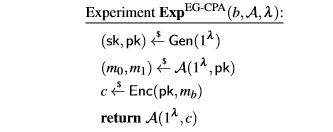
·El Gamal是正确的, 包括乘法同态, 而我们对指数空间中的账户余额进行加密, 从而使同态加法.

· 构造1采用正确的 NIZKs, 只尝试证明正确的陈述.

为了证明易失性，我们为决定性Diffie-Hellman（DDH）问题提供硬度降低。 我们通过一系列的削减来实现这一点。 首先我们考虑以下CPA安全性的经典定义：如果对于所有PPT对手，密码系统（Gen，Enc，Dec）是CPA安全的，那么存在可忽略的函数negl就如：

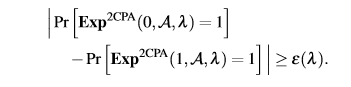


其中ExpCPA(b, A, λ )被定义为

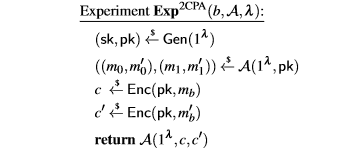


众所周知, El Solidus使用的是 DDH-硬组中的 CPA 安全。我们进一步定义了双 CPA 安全, 我们将使用它来证明我们的 PVORM 结构的易失性。

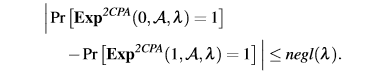
定义 2 (双精度注册安全)。一个密码体制) , (Gen,Enc,Dec)是双 CPA 安全如果为所有 PPT 对手 a 有一个可忽略的函数negl , 这样



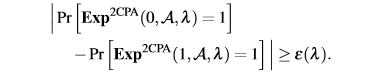
其中**Exp**2CPA(0*,* A*, λ* )被定义为

我们现在证明了一个混合的论点, 任何公钥的密码体制, 是 cpa 安全 (例如, El) 是双注册安全。

**引理 1 (双-CPA 安全)。**让(Gen, Enc, Dec) 是一个注册表安全的公钥密码体制。对于所有有状态 PPT 对手 a, 有一个可忽略的函数 negl(λ) , 这样

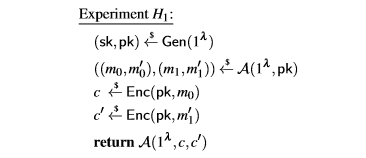


证明. 假定存在某些冲突A和不可ε (λ) 这样



我们现在考虑一组混合实验。让H0 =

Exp2CPA(0, A, λ ), H2 = Exp2CPA(1, A, λ ),以及



请注意，我们加密m0（如H0）和mr（如H2）。 通过标准的混合变元A，在区分H0和H1之间或H1和H2之间必须至少具有ε（λ）/ 2。

我们现在构建一个对手B来打破（Gen，Enc，Dec）的CPA安全。 在输入（1λ，pk）上，B首先运行A得到。 然后随机挑选一个i← - {$ 0,1}。 我们分开处理这些情况。

· i = 0: 在这种情况下B输出) (m0,m1。收到质询后c计算c0 ←−$ Enc, 提交

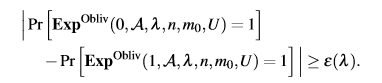
(1λ ,c ,c 0 ) 到 并返回结果.

· i = 1: 在这种情况下B输出. 在收到挑战c(0, 它计算c ←−$ Enc pk, ) 并提交 (1λ) ,c,c 一个并返回结果.

在第一种情况下, 如果c加密m0 , 则这正是实验H1 , 如果c加密m1, 这是实验h2。对于第二种情况, B 也同样生成了实验h0或h1.B将在何时完全成功A成功。由于在这些实验之一中具有至少ε( λ)/2 的优势, 并且B 随机选择要运行的实验,必须是B成功的情况下,至少有ε(λ)/4, 这是不可。然而, 根据假设, (GenEnc,,12月是 CPA 安全的, 因此这与我们的假设相矛盾, 即) 存在。因此 (Gen,Enc,12月) 是双注册安全的。

定理 3 (PVORM 遗忘)。构造1是易失的,在 ROM 中假定为 DDH-硬组.

证明。 假设存在一些PPT攻击者A和不可忽略的ε（λ）这样的矛盾



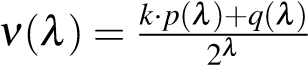
我们现在构造一个对手B这将打破游戏 Exp2CPA, 如引理1所定义的那样, 用于El Gamal.

首先, 我们认为A不能仅基于观察在电路ORAM结构中触及的数据块的模式而区分。正如王、Chan 和施正荣 [46] 所指出的, 每个访问都首先包括访问一个独立于所有以前访问的一致随机路径, 然后沿从访问中独立选择的两条路径进行逐出。因此, 只能希望通过强制隐藏溢出来区分这种方式。王、陈、石还注意到, 即使是 worst-case 的访问模式, 藏匿溢出的概率也微不足道。因此,通过观察电路ORAM访问结构, 可获得最微不足道的优势.

这意味着A必须破坏 NIZK 的语义安全或零知识属性。现在, 我们假定(将在大多数p中 ) 对 PVORM oracle 进行一些多项式p的查询。利用此, 我们构建了一系列的混合分布H0,...,p()+ 修改 ExpObliv oracle 的工作方式.

在混合型H0中,O完全按O 1 , sk ,C 。在H1中, O 的操作方式相同, 但它利用了我们在 ROM 中伪造所有 NIZKs 的事实。对于Hi 与i ≥ 1, 在输入 (u0,u1) 从 A, oracle 应用更新u1作为H1中的第一个i1查询, 之后它应用 u 0而.虽然这可能会导致无效的更新, 但新的 oracle 不会检查有效性并使用伪造的校样来应用更新。由于证明是伪造的, 它将始终成功地作出这 (伪造) 更新。因为, 根据游戏的定义,不能依赖提交无效更新来区分, 这一点根本不能提高优势.

因为我们在 ROM 中工作, 所有 NIZKs 都是 gsp,A在h0和h1中接收相同的视图。每当 PVORM oracle 需要生成一个证据时, 它就会首先选择随机挑战c和响应。然后, 它计算承诺 com, 以确保该元组来自正确的分发, 并修改的随机 oracle, 以便它在查询 com. 上的 oracle 时接收c 。只要未在 com 上对随机 oracle 进行过查询, 此策略就能正常工作并生成与H1相同的分布。

如果存在冲突--在 com 上查询随机 oracle--那么实验H1就简单地中止。幸运的是, 这发生的概率微乎其微。具体地说, 使大多数q (λ) 独立查询对其随机 oracle 进行一些多项式q, 而 O则必须伪造一些每个 PVORM 更新的校样的常量k数。这将冲突的概率限制为, 这是一个可忽略的函数。

我们可以将同一个参数应用于Hp (λ)+1 和对应于的 (未命名的) 混合O 0 , sk ,C 有真正的证据。因此 可以区分h1和h(,)+1 ε (λ) −2ν (λ)。因此, 由一个标准的混合参数, 必须有一定的i ∈ [ 1 , p(,)] 可以区分Hi 和hi+1 有优势的至少。这也是不可。为了简单起见, 我们将表示此优势ε0(λ ).

下一步我们记住, 密钥只用于在更新中生成 NIZKs, 这意味着只有公钥的对手可以运行A使用生成任何H1的 oracle ,...,Hp (λ)+1 . B正是这样的对手.

在输入时(1λ,pk), B首先猜测一个一致地随机i ∈ [1, p(λ)] 然后运行A. B然后处理的A的 PVORM oracle 查询如下所示。对于第一个i − 1 查询, (u0,u1) B将 u1与伪造校样一起应用, 如在hi 和hi、+1。因为更新仅用于校样, 并且 B 是伪造校样, 所以它可以正确地执行余下的更新, 只有 sk. 撤回更新u包含两个明文: 帐户 ID和事务值 $v。因此, 为了生成其所选的明文对, B 输出为 PVORM oracle 查询指定的更新。接收到挑战对的 ciphertexts e ) , = (cid,v 使用更新密文 (和伪造校样) 执行其余更新。对于所有未来的 PVORM oracle 查询, 请在ith 之后, + B 使用更新请求u0-在两个hi 和hi1。当 以输出终止时, B 将输出相同的值。

我们现在声称B在上面定义的Exp2CPA实验中具有不可忽略的优势。 在概率至少为1的情况下，B将选择其中A具有区分Hi和Hi + 1的不可忽略的优势εr（λ）的一些i。 如果B收到u1的挑战加密，那么A正在玩Hi中的游戏。 同样，如果B用u0加密来挑战，那么A就会看到实际的分布Hi + 1。 在这两种情况下，B都会在A确实输出正确的值。 这意味着B必须有利于至少εr（λ），这是不可忽略的。

通过假设，我们正在与一个DDH硬组合并使用El Gamal作为我们的密码系统。因此，我们的密码系统是CPA安全的，所以根据引理1，不存在这样的B。 这与我们对A存在的假设相矛盾，因此建筑1必须是不经意的PVORM。

**定理 4 (PVORM 公共可验证)。**构造1是可公开验证的在 ROM 中.

证明此结果直接来自于以下事实: 我们的更新规范包括每个操作的证明以及一个范围证明。根据定义, 只需验证更新生成的所有 NIZKs。因此, 如果一个对手能够愚弄, 它必须能够伪造 (至少) 一个由更新产生的证据。

假设存在一些PPT敌手A和不可忽略的ε（λ）这样的矛盾比如



我们注意到更新产生三种类型的校样。因此, 我们构建了三新的 PPT 对手 bR, B, b 在 el ciphertexts 上伪造范围证明, 明文等价证明, 并分别证明正确的 el。它们的操作如下：

· B R : 在输入时(pk,sk), B R 运行A并通过关联的 ciphertexts 输出结果范围证明。

· B E : 在输入时(pk,sk), B E 运行A并输出产生的明文等效证明和关联 ciphertexts。

· B S : 在输入时(pk,sk), B S 运行A, 从输出中选取一个一致的随机 El 伪交换证明, 并输出该证明和相关的 ciphertexts。

每当A伪造一个范围证明或一个明文等价证明,B R 或B E 分别成功。对于 B, 对给定的 PVORM 配置 (树深度、存储桶大小和隐藏大小), 更新执行的 el . 正确性证明, BS 将以恒定的概率成功。

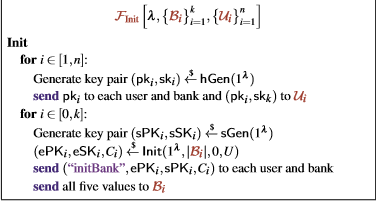
通过检查更新规范和标准混合参数,A必须成功地锻造至少一种具有不可概率的证明类型, 因此它是一个B R , B E 和B S 必须以忽略的概率成功。正如我们在附录 A 中所描述的, 先前的工作表明, 每个相关的证据在 ROM 中都有微不足道的正确性错误。因此, 没有这样 的对手可以存在, 因此Solidus PVORM 构造在 ROM 中是可公开验证的。

**C Solidus 安全性证明**

为了证明Solidus的安全性, 我们证明了我们可以安全地模拟F Sol 在具有可信初始值设定项、理想分类帐和实际应用程序层的混合世界中。我们描述图9中的受信任初始值设定项, 以及图10中理想的分类帐F分类帐 。对于银行级签名, 我们使用 Schnorr 签名 [14, 42], 我们表示由 (sGen,签名,sVer).

我们假定协议中没有直接指定的几个简单的行为。首先, 每个诚实的银行一次只能有一个未决交易。这意味着, 它将不批准一个请求 (作为发送或接收银行), 而有另一个交易, 它已批准尚未清除。在 F分类帐-混合体系中, 这是在 "波特Sol中编纂的, 但我们只是在理想的世界中假设这个属性。第二, 我们假定一个诚实的银行会在收到交易批准请求后立即回复。它可以批准或中止交易, 但它会以某种方式回复。请注意, 一个诚实的银行可能会中止它已经批准的交易, 以便保持可用性。最后, 我们假设, 对于一个诚实的银行, 当一个断言失败时, 银行的行为就像它未能处理的消息完全没有收到。

F分类帐使用 VerTxn 函数验证完全事务。过帐到 F分类帐 包含的交易记录

事务请求、每个银行的 PVORM 更新以及如图11所述的一组其他校样。验证所有校样都需要此信息以及每个银行以前的 PVORM 状态。因此 VerTxn(BBtxn*,,, ,Cr*) 在前面的 PVORM 状态下执行此验证cs 和cr 并返回结果。请注意, 我们使用表示法分类帐[B*,*B] 根据分类帐的当前状态, 最近的 PVORM 状态为B和 b*r*

我们不详细描述如何实现F Init 或F分类帐。相反, 我们注意到 FInit 可以相对简单地使用现有的 PKI 系统来实现。一旦公钥被分发, 它就只会通过正确初始化的相关证明来分发初始 PVORM 状态.

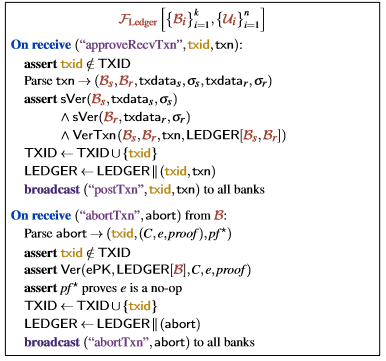
实现F分类帐, 修改各种现有的协商一致算法以执行必要的验证很简单。对于集中式分类帐, 验证是微不足道的。对于分布式协商一致, 当一个节点接收到一个建议的事务时, 它将验证所有的证据并在它们无法验证时忽略该消息。对于拜占庭式容错算法, 这只是一个错误的消息, 共识已经可以处理。只有当银行不会提交无效的交易时, 才能使用不能容忍拜占庭式故障的协商一致算法, 在这种情况下, 这种验证没有影响.

为了简单起见, 我们从证据中省略了资产公证人。添加它们只需要很小的修改。初始化必须公开分发资产公证人身份, F分类帐 必须检查有效的资产公证签名, 而 "波特Sol必须正确显示资产公证身份.

图11描述了F分类帐-混合协议波特Sol用于Solidus。我们通过在 F Sol 世界中构造一个模拟器来证明该协议的安全性 。

**定理1。***Solidus协议Sol满足定义 1, 假设 ROM 中有一个 DDH 硬组.*

*证明.* 我们证明了理想 S*,*Z (*λ*) 和混合*,*Z (*λ*) 使用混合序列是无法区分的。如果是可忽略的, 则概率是可*忽略*的安全参数*λ*的函数。



我们定义混合动力*H*0 *,...,H* 7 。*H*0是 F分类帐-混合世界, 是 一个 "虚拟" 模拟器, 通过不变的传递所有消息。*H*1允许 S 模拟 F分类帐。*H*2将使用伪造和*H*3的诚实方生成的所有校样替换为任意值替换请求和 PVORMs 的内容。在*H*4 S中模拟受信任的初始值设定项并控制所有键。*H*5隔离的事务 id 集和*H*6将任何无效消息从 A中删除。最后, *H*7等效于理想的执行。

混合*H*0包含在之间传递消息的虚拟模拟器A和诚实的党不变。这与F分类帐混合体系相同。

混合*h*1与*h*0相同, 但该S维护其自己的模拟副本F分类帐该行为与F分类帐除初始化之外, 它不模拟。在初始化过程中, 的将传递由 F分类帐 到,而不进行修改的实际值。所有其他操作都被忠实地模拟。我们注意到所有 non-initialization 操作只需要公共信息 (包括公钥)。当一个诚实的银行发布到 F分类帐, 将邮件复制到其自己的副本, 并且当 A 发布到 F , S 首先模拟其副本上的行为, 如果该帖子被接受, 则将该帖子转发到真正的F分类帐.

从所有帖子到F分类帐要么以静默方式退出, 要么全部广播到所有银行,S对副本的忠实模拟将导致与实际执行相同的视图。

混合*h*2在*h*1中继续进行, 除非每次S接收由诚实方构造的任何校样或签名-作为请求、PVORM 更新或的一部分"postTxn"邮件来自F分类帐-它存储真实的校样和签名, 并用伪造替换它们.S将伪造的发送到A(或模拟 F分类帐), 如果包含这些校样的邮件将被发送回一个诚实的当事方 (或转发到真正的 F分类帐), S 将原始 (实际) 校样和签名放回原位。

请注意, 此伪造和替换仅适用于由诚实方构建的特定校样和签名。来自包含来自的来自控制方的证明和签名的诚实方的消息, 例如在最终事务中包含的来自受控用户的请求签名--只有被替换的诚实的署名和证明。由计算的值 完全保留在机智中

由于系统中的所有校样都是 NIZKs,S可以伪造证明A将接受和A仍无法伪造具有不可概率的证明。由于从*H*1中更改的唯一的东西是这些伪造的校样, 所以*h*1和*h*2在计算上是无法区分的。

混合*h*3非常类似于*h*2, 但S还会替换诚实方密钥 (包括 PVORM 值) 下由诚实的各方生成的所有加密的值.S将这些值替换为在同一键下加密的随机选择的值。再次, 它保存真实的价值和真实的证明时, 与诚实的各方沟通, 但它使用随机值与 。由于 S 只替换未生成且在不知道密钥的公钥下加密的值, 因此加密方案的语义安全性保证*H*3与*h*2之间无法区分。在*H*2中, 校样并不表示它们已经伪造 (对于实际值), 因此它们在*H*3中保持伪造 (对于随机值)。

混合*h*4不同于*h*3中的S现在在中模拟初始化F Init 。它从正确的发行版中生成假密钥和 PVORMs, 并将其发送到,而不是由 F Init生成的那些。由编写的任何加密值都将在S已知道密钥的新 (假) 密钥下进行加密 , 并且任何值都要由A 和由一个诚实的方编写将在真正的F初始下的S的密钥下进行加密。在这两种情况下, 的都可以解密密文并在另一组密钥下重新明文, 然后将一条真诚的消息传递给A 或 A’的消息给一个诚实的一方。对于由创建的签名和校样,

对于诚实方密钥下的加密, 以及由诚实方创建的证明和签名,S已将那些在*H*3中分别替换为随机值和伪造, 因此它只在新 (假) 键下进行相同的处理。

以这种方式查看的所有值、校样和签名Ain *h*4与*h*3中的相同, 但使用不同的加密/签名密钥和不同的随机性。由的生成的所有加密、校样和签名都 类似于一个诚实的方, 但具有不同的随机性。因为钥匙和随机性从确切地被选择的原始的发行忠实地, *H*3和*h*4是相同地分布的。

混合*h*5继续进行*h*4, 但S分隔使用的事务 idA从那些诚实的党使用。每当新请求 来自具有事务 ID的时 , txid将生成一个新的唯一 txidF ,以将交易与诚实的当事人联系起来。每当具有以前看不见的事务 ID txidF 来自一个诚实的一方 (或 F,分类帐), S 在转至 (或模拟 F分类帐之前) 生成新的唯一txid 。如果对于任何一个方向的传入消息, 的以前都看到过该 id, 则在另一组中必须有一个关联的 id, 因此它只是使用它.

由于只有事务 id 已更改, 并且使用相同的方法从旧 id 独立绘制新 id, 所以*H*4和*H*5的分布相同。

混合*h*6与*h*5除外S验证生成的所有校样和签名A在所有邮件上。如果任何证据或签名无法验证,的将 删除该消息, 但不转发该邮件。由于在进行任何其他处理之前, 所有校样都在Sol (由接收方或 F分类帐) 中验证,并且 Z 指示如果断言失败, 则诚实的党的行为就好像关联的消息从未到达过, 这不会更改接收的任何消息 或任何诚实方的行为。同样, *H*6将所有包含已过帐到 F分类帐的事务 id 的邮件都删除, 而诚实的当事方也同样会删除这些消息。通过模拟使用的 NIZKs 的可靠性, 具有可忽略的伪造证据的可能性, 因此, 通过将被忽略的消息的概率可以忽略不计。因此, *H*5和*H*6在计算上是无法区分的。

混合*H*7是最复杂的步骤, 因为我们现在用替换所有的诚实方通信F Sol 。现在, 我们将描述的 在 h 6中向一个诚实的方发送消息时所做的事情, 以及每当它从 F Sol 中收到一条消息时, *H*7.：

· 当S将发送"requestTxn"向诚实银行提出请求B代表已泄露的用户U *s* 在*H*6中,S改为解密提供的值A获取纯文本值$*v* 和接收用户U *r* 并发送(*,*) "requestTxn"*,*U*r*to F Sol 代表U *s* 。与其创建它自己的txidF 以链接到该事务的txid,,它将使用由 FSol 返回的那个。

· 时S将发送"approveSendTxn"消息到和诚实的银行在*H*6中, 它首先检查是否有关联的txid F 来自F Sol , 或者如果邮件来自自发A。在第一种情况下, 它发送 ("approveSendTxn"*,*txidf) f.在第二种情况下, 它首先解密包含在事务数据中的请求, 必须是从受损害的用户 U 在受损害的银行 B,否则该请求将通过F Sol 或校样无法验证, *H*6已将其删除。然后, 它将关联的 "requestTxn" 消息提交到 FSol U 。接收到关联的 *,* txidF ("要求"*,*txidf*,*) *,v*, ) ("approveSendTxn"*,*txidf fSol .时S将发送"approveRecvTxn"消息到真实F分类帐(经过模拟的一个) 后, 它再次检查关联的txid F 来自F Sol 。如果没有找到, 则事务必须完全由受损害的实体执行, 原因与上述相同。在这种情况下,的 对事务详细信息进行解密, 并在 FSol 上执行整个事务。如果txid F 找到, 并S已看到"必需"响应来自F Sol 但不是"aprv"消息, 则必须是两个银行都受到损害的情况。与上面一样, ) 的按顺序完成事,首先发送 ("approveSendTxn"*,*txid F ("approveRecvTxn"*,*txidF )。

最后, 如果txid F 找到, 并S已看到"aprv"邮件来自F Sol fortxid F , 则它只发送("approveRecvTxn"*,*txidF )。

· 当S将发送"abortTxn"真正的消息F分类帐, 它将再次检查是否有关联的txid F 。如果有, 它将发送 ) ("abortTxn"*,*txidf f如果没有, 它将生成一个随机的 txid 并发送 ("abortTxn"*,*txid) F.。

请注意, 此新的可能性微乎其微txid将与现有事务 ID 冲突, 并且不会接收中止, 但除非概率很低, 否则将为不存在的事务创建中止。

· 我们处理S接收("必需"*,*F*,*txid*,* 在两种情况下 *,*$*v*) 从 FSol

1. IfB *r* 是诚实的, 那么S在接收到有效时, 它在*H*6中的作用

("requestTxn" *,* txid F *,* ePK*s ,cv ,cr ,σ* ) 从 U*s*, 在这种情况下, 它可以解密 U和 $的身份 , 但不能是接收用户的标识.

2. IfB *r* 已失密, 而S将转发一个"requestTxn"消息在*H*6中, 它没有足够的信息来正确地创建该请求的细节。要获取该信息, 的立即答复 FSol 与 ("approveSendTxn"txid F)

·当S接收(*,,*"aprv"txidF*, ,*) $*v*从 F, 我们再次有三例：

1. IfB *s* 已失密, 那么我们必须在上面的情况下2。因此,的现在有足够的信息来创建完整的"requestTxn" 消息, 就像在*H*6中一样, 因此它会这样做并将请求提交到 .

2. IfB *s* 是诚实的, 但最初请求此事务的用户U *s* 不是, 则必须有一些txid关联txid F 和关联的请求.S因此可以制造一个"approveSendTxn"要提交的邮件A。如在*H*6中, s使用由U创建的值所存储的请求和由诚实的 B 创建的伪造值 *s* .

3. IfB *s* 和发送用户U *s* 都是诚实的, 那么S必须创建新的唯一性txid并创建"approveSendTxn"消息为*H*6。请注意, 的的值可以在*H*6中进行解密, 其身份为U*r* 和 $*v*, 因此它将加密更正这些值和伪造其他值。

· 当S接收 ("postTxn" *,*Ftxid*,*) 从 F Sol, 因为此证明不处理资产公证员, 我们可以假设 p. *r* 都是银行。有三例需要考虑.

首先我们考虑最简单的情况: 当P *r* 是一个受攻击的银行。在这种情况下, 事务将只通过 FSol 后 S 成功地将其发布到 (模拟) F分类帐。因此, 没有什么可做的.

下一步我们考虑的情况下P *s* 是一个受攻击的银行, 但P *r* 是诚实的。这里txidF 必须对应 txid中的待处理事务, P *r* 。为了使事务在FSol 中由发件人批准, 必须已接收并验证 ("signRecvTxn"*,* *,* txidtxdata) 在这一点上, S 更新 P*r*的模拟 PVORM 带有随机值和伪造校样 (如*H*6), 并将整个事务发布到F分类帐。我们注意到, 不能已将事务提交到 F分类帐 带 ID txidA 因为诚实的银行响应立即, 因此, 这必须是响应批准发送事务和*H*6将删除该消息, 如果 txid已过帐到 F分类帐.

最后, 我们考虑的情况下P *s* 和P *r* 都是诚实的。在这种情况下, S将随机更新各自的 PVORMs 并伪造所有相关的校样。如果 txidF 已经对应某些 txid, 这意味着请求的用户受到了威胁, S 只使用请求.否则, S将选择一个新的唯一 txid, 并创建一个请求规范 (再次使用随机值和伪造校样)。然后, 它将结果发布到模拟的 F分类帐。我们注意到, 这恰恰是将被过帐到模拟 F分类帐 在*H*6中的值。

· 当S接收() "abortTxn"*,*txidF*,*B来自F Sol , 则首先检查B已泄露。如果是这样, 则在将中止发送到 FSol 并没有任何可做的事情之后, 这必须是响应。如果没有, 则的将检查是否有已知的txid已链接到 txid , 并生成一个新的唯一 txid否则。然后, 它使用随机值和伪造校样生成中止操作, 如在*H*6中, 并将其发布到 F分类帐。它还清除了 B 的模拟挂起事务 (仅当 txid已存在时才发生)。

因此, 我们看到, 每个混合的计算无法区分从下一个, *H*0对应于F分类帐-混合世界, *H*7对应于理想的世界。因此,Sol实现了所需的安全性。

**D 变种**

现在, 我们在基于不同体系结构基元的Solidus系统上呈现三变种。它们提供了我们认为相关的不同的保证和特征。

**D1 zk-SNARK PVORM**

虽然 gsp 是非常有效的建设, 他们可以是相当大, 昂贵的验证。在证明的大小或验证时间比生成时间更重要的情况下, zk-虫子提供了一个好的选择。虽然我们可以使用 zk-虫子来实现4节和附录 B 中描述的电路 ORAM-based PVORM, 但大量的 reencryptions 会导致非常昂贵的证明, 即使我们使用对称密钥原语。相反, 在7.3 节, 我们评估了一个不同的结构, 基于 Merkle 树, 这是更有效的 zkSNARKs 比使用电路ORAM。

在这个 zk 斯纳克友好的 PVORM 中, 每个帐户 (或其他数据) 都存储在标准的 Merkle 树的叶子上。树的根, 但没有其他节点, 将过帐到分类帐。在收到已签名的更新请求时, 银行会将一个帐户更新为有效值, 并相应地修改 Merkle 树。然后, 它生成一个 zkSNARK, 它正确地执行了更新, 并正确地验证了请求的签名。新的 Merkle 树的根是新的 PVORM 状态和 zk 斯纳克是证明。

我们在表1中的评估显示了以128位安全级别进行的单个银行更新的性能, 使用 libsnark作为计算 zk 斯纳克校样的后端。Merkle 树的深度为 15, 使 PVORM 的容量为 215 (与我们的 GSP 测试相同)。我们的实现包括 zk-SNARKoptimized SHA-256 电路的 Merkle 树, 和优化电路的 RSA-3072 加密 (RSAESPKCS1-v1 5) 和签名 (RSASSA-PKCS1-v1 5 与 SHA-256)。我们使用了 pkcs #1 v1.5 基元, 而不是最新的 pkcs #1 v2.2 基元和替代的公钥方案的三原因: 他们产生更便宜的 zk-SNARK电路, 他们仍然在实践中使用, 他们提供了一个保守 (即竞争) 比较点的 gsp。

从我们在7节的结果中可以看出, 这一构造涉及的证明生成时间比 gsp 的要慢两个数量级。对于 singlethreaded 执行, 或具有保守的3桶参数化和 215帐户的 GSP PVORM, 需要0.4 秒才能生成校样, 而 zk 斯纳克 PVORM 需要在同一台机器上使用65.5 秒和等价设置 (亚马逊 EC2c4.8xlarge 实例)。对于单线程执行, 我们的 GSP PVORM 需要0.4 秒以在 c4.8xlarge 亚马逊 EC2 实例上使用保守的3桶参数化和 215帐户生成证明。反之, zk-斯纳克 PVORM 的验证时间大约是两个数量级快 (0.0065 秒与0.56 秒), 证明是相当紧凑 (288 字节)。

在Solidus中使用时, zk-斯纳克 PVORM 结构具有明显的缺点, 即分类帐不包含每个用户的帐户余额, 即使是以加密的形式。要计算用户的余额, 审核员需要分析事务 ciphertexts, 解密并执行所有操作。然而, 为了减少实际的费用, 银行可以定期检查差额。具体地说, 它可能会提交 Merkle 树叶子的加密版本, 并证明加密与发布的 Merkle 树摘要使用另一个 zk-SNARK证明是一致的。这样的证明是相当昂贵的构造, 而且只能定期进行, 例如, 每天一次, 而不会影响系统的吞吐量。但是, 由于交易伴随着 ciphertexts, 审计员可以从检查点开始, 然后解密所有后续交易, 以了解经常账户余额。

当然, 在Solidus的应用中, 证明生成时间更重要, 在我们与区块链行业技术专家的讨论中, zk-SNARK和可信安装的工程复杂性使它们比 gsp 更不可行。但 zk-SNARK提供了一个有趣的替代结构, 并说明了什么最终可能是一个有价值的点, 在 PVORM 设计空间。

D2 使用守信任的信件

使用英特尔软件防护扩展 (新交所), 可以构建一个更高效的 PVORM。新交所提供了一组新的指令, 允许在*飞地*内执行应用程序, 从而保护应用程序的控制流完整性和保密性, 使其不受恶意操作系统的攻击。新交所还启用了*认证*的生成, 它向远程方证明了飞地正在运行特定应用程序 (标识为其生成内存的哈希)。

为了减少认证的开销, 飞地可以生成签名密钥对并证明公钥的完整性。然后, 它可以通过简单地签署一个断言, 它知道该语句的见证, 从而生成等价的 NIZK。信托在新交所, 然后转化为信任的应用程序, 从而其断言。验证断言只需要单个数字签名验证。

使用 SGX-based 的方法, 我们可以建立一个非常快的 PVORM。我们用对称密钥加密和所有 NIZKs 与新交所签署的断言来替换公钥加密。我们甚至可以使用只写

ORAM进一步提高性能。此外, 在*密封-玻璃校样*(小额赠款协议) 模型 [45] 中构建的 PVORM 提供了针对任意强的侧通道攻击的安全性, 前提是密钥签名键仍受保护-例如通过使用侧 channelresistant加密库。

虽然有几个复杂的问题有待解决 (例如, 需要在不同的主机上共享密钥以防失败), 但我们认为这种方法非常实用--尽管在英特尔 (强烈) 的信任的前提下, 以及它的实施新交所。

**D3 Pedersen承诺的使用**

Solidus的一个重要特性是可, 它通过在分类账上加密所有帐户余额而得到极大的帮助。然而, 许多金融公司和监管机构都小心翼翼地将这些信息包括在内, 即使是以加密的形式。虽然我们认为它会降低功能显着省略这些加密, 它不是特别困难。

而不是在分类账上包括加密的余额, 银行可能代替代表 PVORM 的元素作为Pedersen承诺。不同于 El ciphertexts, Pedersen承诺是完全隐藏和计算绑定。为了实现这一点, 银行需要为每项承诺保留证人, 包括帐户余额和随机因素。然后, 银行可以向审计员披露这一证人, 以便证明帐户余额, 附录 A 中的证明方案只需要稍加修改, 以证明已知证人的资料。