走近BTC：理解BitVM所需的背景知识

作者：Nickqiao & Faust & Shew Wang，极客web3

摘要：近期Delphi Digital发布了题为《The Dawn of Bitcoin Programmability: Paving the Way for Rollups 》的比特币二层相关技术研报，系统的梳理了和比特币Rollup有关的核心概念，如BitVM全家桶、OP\_CAT和Covenant限制条款、比特币生态DA层、桥以及Bitlayer、Citrea、Yona、Bob等四大采用BitVM的比特币二层。

该研报虽然大体展示了比特币二层技术的大致图景，但整体比较泛泛而缺乏细节描述，让人似懂非懂。极客web3在Delphi研报基础上进行了展开式的深入挖掘，尝试让更多人系统的理解BitVM等技术。

我们将与Bitlayer研究团队及BitVM中文社区共同开展一个名为“走近BTC”的系列专栏，长期围绕BitVM、OP\_CAT和比特币跨链桥等重点话题进行科普，致力于为更多人祛魅比特币二层相关技术，帮更多爱好者铺平道路。



正文：几个月前，ZeroSync负责人Robin Linus发布了名为《BitVM: Compute Anything on Bitcoin》的文章，正式提出了BitVM的概念，推动了比特币二层技术的进展。可以说这是比特币生态最具革命性的创新之一，引爆了整个比特币二层生态，吸引了如Bitlayer、Citrea、BOB等明星项目的参与，为整个市场带来了生机。

之后，更多研究人员参与改进了BitVM，先后推出了BitVM1、BitVM2、BitVMX、BitSNARK等不同的迭代版本。其大致情况如下所示：

Robin Linus于去年最先提出的BitVM实现白皮书，就是基于虚构逻辑门电路的BitVM实现方案，被称为BitVM0；

Robin Linus在后面几次演讲和采访中，又非正式的介绍了基于虚构CPU的BitVM方案（称为BitVM1），类似于Optimism的欺诈证明系统Cannon，可以用比特币脚本在链下模拟出一个通用CPU的效果。

Robin Linus还提出了BitVM2，一个Permissionless的单步非交互式欺诈证明协议。

Rootstock Labs和Fairgate Labs的成员发布了BitVMX白皮书，与BitVM1类似，他们希望通过比特币脚本模拟出通用CPU的效果（在链下）。

目前BitVM相关开发者生态的建设日渐明朗，周边工具的迭代完善也已肉眼可见，相比于去年，如今的BitVM生态已经从最初的“空中楼阁”变得“依稀可见”，这也吸引了越来越多的开发者和VC争相涌入比特币生态。

但对于大多数人而言，要理解BitVM和比特币二层相关的技术名词绝非易事，因为你要先对其周边的基础知识有系统性的理解，尤其是比特币脚本和Taproot等背景知识。目前网上已有的参考资料要么篇幅太长废话连篇，要么解释的不够透彻让人似懂非懂。我们致力于解决上述问题，力求以尽可能清晰的语言，帮助更多人理解比特币二层的周边知识，对BitVM体系建立起系统性认知。



MATT和承诺：BitVM的基础思想

首先我们要强调，BitVM的基础思想是MATT，含义是Merkleize All The Things，主要指通过Merkle Tree这种树状的数据存储结构来展示复杂的程序执行过程，设法让比特币Native的验证欺诈证明。

MATT虽然可以表达出一段复杂程序及其数据处理痕迹，但不会直接在BTC链上发布这些数据，因为这些数据的总体规模非常庞大。采用MATT的方案只在链下的Merkle树中存储数据，只把Merkle树最顶部的摘要（Merkle Root）发布到链上。这棵Merkle树主要包含三大核心内容：

· 智能合约脚本代码

· 合约所需的数据

· 合约执行中留下的痕迹（智能合约在EVM等虚拟机中执行时对内存、CPU寄存器产生的变更记录）



（一个简单的Merkle Tree默克尔树示意图 其Merkle Root是由图中底部的8个数据片段经过多层hash计算得到的）

MATT方案下，只有尺寸极小的Merkle Root存储在链上，Merkle Tree包含的完整数据集存储在链下，这用到了一种被称为“承诺”的思路。这里解释下什么是“承诺”（Commitment）。

承诺类似于一种简洁化的声明，我们可以把它理解为一大批数据压缩后得到的“指纹”。一般而言，在链上发布“承诺”的人会声称，某些存放在链下的数据是准确无误的，这些链下数据要对应一个简洁化的声明，这个声明就是“承诺”。

在某些时候，数据的hash可以作为对数据本身的“承诺”，其他的承诺方案还有KZG承诺或Merkle Tree等。在Layer2惯用的欺诈证明协议中，数据发布者会在链下发布完整数据集，在链上发布数据集的承诺。如果有人发现链下的数据集中存在无效数据，就会针对链上的数据承诺进行挑战。

通过承诺（Commitment），二层能够把大量数据压缩处理，只在比特币链上发布其“承诺”。当然，还要保证发布在链下的完整数据集可以被外界观测到。



目前几大BitVM方案如BitVM0、BitVM1、BitVM2和BitVMX，基本都采用了类似的抽象结构：

1.程序分解和承诺：首先将复杂的程序分解为大量的、较基础的操作码（编译），然后把这些操作码在具体执行时产生的痕迹记录下来（说白了就是一段程序跑在CPU和内存中时，整个的状态变化记录，称为Trace）。之后，我们对包括Trace和操作码在内的所有数据进行整理，组织成一个数据集，然后生成该数据集的承诺。

具体的承诺方案可以有多种形式，如：Merkle树、PIOPs（各种ZK算法）、哈希函数

2.资产质押和预签名：数据发布者和验证者需要通过预签名的形式，把一定金额的资产锁定在链上，并且会有限制条件。这些条件会针对未来可能发生的情况而针对性的触发，如果数据发布者作恶，验证者可以提交证明把数据发布者的资产拿走

3.数据和承诺发布：数据发布者在链上发布承诺，链下发布完整的数据集，验证者检索数据集并检查是否有任何错误。链下数据集中的每个部分都与链上的承诺有关联性。

4.挑战和惩罚：一旦验证者发现数据发布者提供的数据有错误，它会把这部分数据拿到链上去直接验证（要先把这部分数据切的特别细），这就是欺诈证明的逻辑。如果验证结果显示，数据发布者的确在链下提供了无效数据，它的资产就会被挑战他的验证者拿走。

总结下就是，数据发布者Alice在链下公开二层交易执行过程中产生的所有痕迹，把对应的承诺发布到链上。如果你要证明某部分数据有误，先向比特币节点证明这部分数据和链上的承诺相关联，也就是证明这些数据是Alice本人对外公开的，然后让比特币节点确定这部分数据有错误。

现在我们大致理解了BitVM的整体思路，所有的BitVM变体基本都脱离不了上述范式。那么接下来，让我们开始学习和理解上述流程中用到的一些重要技术，先从最基础的比特币脚本和Taproot以及预签名开始。

什么是Bitcoin Script脚本

比特币相关的知识要比以太坊的更难理解，就连最基础的转账行为都涉及到一系列概念，包括UTXO（未花费的交易输出）、锁定脚本（也称为ScriptPubKey）和解锁脚本（也称为ScriptSig）。我们先对这几个主要概念进行讲解。



（一段比特币脚本代码的示例 由比高级语言更底层的操作码组成）

以太坊的资产表达方式，更像支付宝或者微信，每次转账只是对不同账户的余额做加减法，这种方法是以账户为核心，资产余额只是账户名下的一个数字；比特币的资产表达形式更像黄金，每块黄金（UTXO）都会标记出主人，转账实际上是把旧的UTXO销毁，把新的UTXO产生（主人会变更）。

比特币UTXO包含两个关键字段：

数额，以“聪（satoshi）”为单位（一亿聪为一BTC）；

锁定脚本，也称 “脚本公钥（ScriptPubKey）”，会定义UTXO的解锁条件。

需要注意的是，比特币UTXO的所有权是通过锁定脚本来表达的，如果你要把自己的UTXO转让给Sam，可以发起交易销毁自己的某个UTXO，把新生成的UTXO的解锁条件写为“只有Sam可解锁”。

之后，Sam如果要使用这些比特币，需要提交一个解锁脚本（ScriptSig），在这个解锁脚本中Sam要出示自己的数字签名，证明自己是Sam本人。如果解锁脚本和前述锁定脚本相匹配，Sam就可以解锁并把这些比特币再转给别人。



（解锁脚本要和锁定脚本相匹配才行）

从表现形式的角度看，比特币链上的每笔交易都对应着多个Input和Output，每个Input中要声明自己想解锁的某个UTXO，并提交解锁脚本，解锁并销毁该UTXO；Output中会展示新生成的UTXO信息，对外公示锁定脚本的内容。

比如，在一笔交易的Input中，你证明自己是Sam，把别人给你的多个UTXO解锁，统一销毁，再生成多个新的UTXO并声明让xxx在未来去解锁。



具体而言，在交易的Input数据中，你要声明自己要解锁哪些UTXO，并指出这些UTXO数据的“存储位置”。这里要注意，比特币和以太坊截然不同，以太坊提供了合约账户和EOA账户两种账户来存储数据， 资产余额作为数字，记录在合约账户或EOA账户名下，统一放置在名为“世界状态”的数据库中，转账时直接从“世界状态”中对特定账户进行修改，便于定位到数据的存储位置；

比特币没有世界状态的设计，资产数据分散存储在过往的区块中（就是未解锁的UTXO数据，在每笔交易的OutPut中单独存放）。



如果你想解锁某个UTXO，要说明该UTXO信息存在于过去哪笔交易的Output中，出示这笔交易的ID（就是其hash），让比特币节点去历史记录中寻找。如果要查询某个地址的比特币余额，需要从头遍历所有区块，找出和xx地址关联的未解锁UTXO。

平时用比特币钱包时，可以快速检查某地址拥有的比特币余额，很多时候是因为钱包服务自身通过扫描区块，对所有地址建立了索引，方便我们快速查询。



（当你生成一笔交易声明把自己的UTXO送给别人时，要根据这些UTXO所属的交易hash/ID来标记出该UTXO在比特币历史记录中的位置）

有意思的是，比特币交易的结果是在链下计算完成的，用户在本地设备上生成交易时，就要直接把Input和Output全部创建好，相当于把交易的输出结果计算完了。交易在广播到比特币网络中，被节点验证后才上链。这种“链下计算—链上验证”的模式与以太坊是完全不同的，在以太坊上，你只需要提供交易输入参数，交易结果由以太坊节点计算并输出。

此外，UTXO的锁定脚本（Locking Script）是可以自定义的，你可以把UTXO设定为“某个比特币地址的主人可解锁”，该地址的主人需要提供数字签名和公钥（P2PKH）。而在Pay-to-Script-Hash（P2SH）交易类型中，你可以在UTXO锁定脚本中添加一个Script Hash，谁能提交这个Hash对应的脚本原像，并满足该脚本原像中预设的条件，就可以解锁UTXO。BitVM所依赖的Taproot脚本，用到了类似于P2SH的特性。

比特币脚本怎么触发

这里我们先以P2PKH为案例介绍比特币脚本的触发方式，只有理解了其触发方式才能理解更为复杂的Taproot和BitVM。P2PKH全称“Pay to Public Key Hash”，在这种方案下，UTXO的锁定脚本中会设置一个公钥hash，解锁时需要提交对应该hash的公钥，这和常规的比特币转账思路基本一致。

此时，比特币节点要确定解锁脚本中的公钥，和锁定脚本中指定的公钥hash能对上号，也就是说，要确定解锁人提交的“钥匙”和UTXO预设的“锁”彼此匹配。

进一步说，P2PKH方案下，比特币节点收到交易后，会将用户给出的解锁脚本ScriptSig，与要解锁的UTXO的锁定脚本ScriptPubkey拼接到一起，放在BTC脚本的执行环境内执行。下图给出执行前的拼接结果:



可能读者并不了解BTC的脚本执行环境，此处我们进行简单介绍。首先，BTC脚本包含两种元素:

数据和操作码。这些数据和操作码会按照从左到右的顺序，依次压入栈内按照指定逻辑来执行，得到最终结果（关于什么是栈 此处不展开详述 读者可以自行Chatgpt）。

以上图为例，左侧是某人上传的解锁脚本ScriptSig，包含他的数字签名和公钥，而右侧的锁定脚本ScriptPubkey中，包含UTXO创建者生成该UTXO时设置的一段操作码和数据（此处我们不需要了解每个操作码的含义，理解个大概即可）。

上图中右侧的锁定脚本中的DUP、HASH160、EQUALVERIFY等操作码，负责把左侧的解锁脚本中携带的Public key取哈希，和锁定脚本中预设的Public key hash做对比，若两者相等，说明解锁脚本中上传的公钥，和锁定脚本中预设的公钥哈希相匹配，这就通过了第一道验证。

但是，有个问题，UTXO锁定脚本的内容其实是在链上公开的，任何人都能观测到其中包含的公钥哈希，谁都可以上传对应的公钥，谎称自己是那个被“钦定”的人。所以在验证完公钥和公钥hash后，还要验证交易发起人是否真是该公钥的实际控制者，这就要对数字签名进行核验。锁定脚本中的CHECKSIG操作码，就是负责验证数字签名的。

总结一下，P2PKH方案下，交易发起人提交的解锁脚本中，包含公钥和数字签名，该公钥要和锁定脚本中指定的公钥哈希匹配，且交易的数字签名正确，满足这些条件才能顺利解锁UTXO。



（这个图是动态的：P2PKH方案下比特币解锁脚本示意图

来源：https://learnmeabitcoin.com/technical/script）

当然，比特币网络中支持多种交易类型，不只有Pay to public key/public key hash，还有P2SH（Pay to Script hash）等，一切取决于UTXO创建时自定义的锁定脚本被设置成什么样。



这里需要注意的是，P2SH方案下，锁定脚本中可以预设一个Script Hash，而解锁脚本需要把Script Hash对应的脚本内容完整提交上来。比特币节点可以执行这段脚本，如果这段脚本里定义了多签验证的逻辑，就可以在比特币链上实现多签钱包的效果。

当然，P2SH方案下，UTXO创建者要让未来解锁UTXO的人事先知道Script Hash对应的脚本内容，只要双方都知道这段Script的内容，那么我们就可以实现比多签更复杂的业务逻辑。

这里要说明一点，比特币链上（区块）并不直接记录哪些UTXO和哪些地址关联，它只记录UTXO可以被哪个公钥哈希/哪个脚本哈希解锁，但我们根据公钥hash/脚本hash可以快速算出对应的地址（钱包界面显示的那一段像乱码的东西）。

我们之所以能在区块浏览器和钱包界面看到xx地址下有xx数额的比特币，是因为区块浏览器和钱包项目方帮你解析了这些数据，会扫描所有区块并根据锁定脚本中声明的公钥hash/脚本hash，计算出对应的“地址”，然后显示出xx地址名下有多少比特币。

隔离见证与Witness

当我们理解了P2SH的思路后，便和BitVM所依赖的Taproot更近一步了。但在此之前，我们要了解一个重要的概念：Witness和隔离见证。

复盘前面讲到的解锁脚本和锁定脚本，以及UTXO解锁流程，会发现一个问题：交易的数字签名包含在解锁脚本中，生成签名时不能把解锁脚本覆盖进去（生成签名用到的参数不能包含签名本身），所以数字签名只能覆盖解锁脚本之外的部分，也就是只能与交易数据的主干部分建立关联，不能完整的覆盖交易数据。

这样一来，就算交易的解锁脚本被中间人稍做手脚，也不会影响到验签结果。比如说，比特币节点或矿池可以在交易的解锁脚本中，塞入其他数据，在不影响验签和交易结果的前提下，使得交易数据发生细微变化，最后算出的交易hash/交易ID也会改变。这被称为交易延展性问题。

这带来的坏处是，如果你打算连续发起多笔交易，并且有次序上的依赖关系（比如，交易3引用了交易2的输出，交易2引用了交易1的输出），那么排后面的交易必然要引用前面交易的ID（hash），矿池或比特币节点等任意中间人可以微调解锁脚本中的内容，使交易上链后的hash与你预期的不一致，那么你预先创建好的多笔有次序关联的交易会失效。

实际上，在DLC桥和BitVM2的方案中，会批量构建有先后次序关联性的交易，所以前面提到的场景并不少见。



简单来说，交易延展性问题是因为，交易的ID/hash在计算时，会把解锁脚本的数据包含进去，而比特币节点等中间人可以微调解锁脚本中的内容， 导致交易ID与用户预期的不符合。其实这是比特币在早期设计时考虑不周留下的历史包袱。

后来推出的隔离见证/SegWit升级，其实就是把交易ID和解锁脚本彻底解耦，计算交易hash时不需要把解锁脚本数据包含进去。遵循SegWit升级的UTXO锁定脚本，会默认在首位设置一个叫“OP\_0”的操作码，充当标记；而对应的解锁脚本，从SigScript更名为了Witness（见证）。



遵循隔离见证规则后，交易延展性问题会被妥善解决，你不需要担心发送给比特币节点的交易数据被微调。当然我们不需要想的太复杂，P2WSH的功能和前面谈到的P2SH并无本质差异，你可以在UTXO锁定脚本中预设一个脚本哈希，等解锁脚本的提交者把hash对应的脚本内容提交到链上并执行。

但如果你要实现的脚本内容特别庞大，包含特别多的代码，通过常规的方法无法把完整的脚本提交到比特币链上（每个区块有大小限制）。那怎么办？这就需要借助Taproot，针对上链的脚本内容进行精简化处理，而BitVM正是基于Taproot构建出的复杂方案。