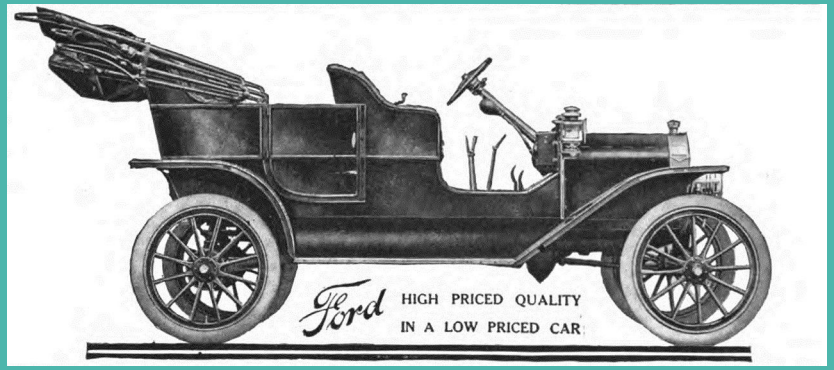
Bitcoin 一层资产协议概述

Cipher @ CELL Studio

古早科技



染色币：EPBOC / Open Assets

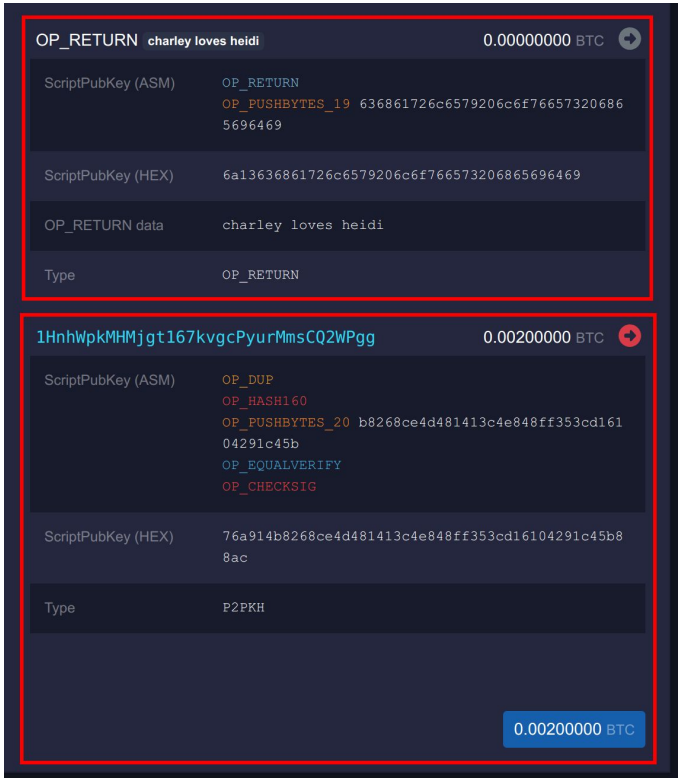
- 最早的 EPBOC 方案采用直接对聪染色，即“一币一聪”，数据保存在 4 字节的 nSequence 中
- Open Assets 之后就采用了信息保存在 OP_RETURN 的方式，对 UTXO 染色，因此摆脱了“一币一聪”的限制
- 使用数字签名确保资产ID不可伪造
- 需要回溯历史交易确保资产的真实性
- 不需要验证所有人的所有交易

Mastercoin / Omnilayer

- Mastercoin 做了史上第一次ICO，融资 5000 个 BTC
- *Mastercoin: A Second-Generation Protocol on the Bitcoin Blockchain* (Vitalik)
- 早期 Mastercoin 将染色信息编码在输出的 UTXO 地址里面，后面切换到 Omnilayer 的方式
- Omnilayer 将交易信息直接以账户操作方式写入 OP_RETURN
- 染色币采用的是 UTXO 模型，余额和操作与 UTXO 绑定；Omnilayer 采用的是账户模型，余额和操作与地址绑定
- 需要验证所有人的所有交易

OP_RETURN 介绍

- 在交易的输出位置出现
- 一笔交易存在至多一个
- 至多 80 字节
- 占用 0 聪，需要支付额外的 fee
- 共识层不做限制，在节点传播侧限制



https://mempool.space/tx/8bae12b5f4c088d940733dcd1455efc6a3a69cf9340e17a981286d3778615684

铭文时代

Ordinals

- 序数理论：为每一个聪定序、定稀有度
- 通过 indexer 追踪每一个定序的聪
- 铭刻：将资产或信息“附加/绑定”到特定聪的过程，等于为它做了个记号
- 铭刻具体的技术操作：将数据放到 taproot 相关的隔离见证区域，一次最多可以存放数百字节
- 铭刻需要 commit & reveal 两笔交易才能将数据写入区块链
- 通过铭刻，可以将图片数据写入 bitcoin，从而实现全链上的 NFT
- 仅创建时需要铭刻，转让和交易都是直接转移聪，不需要铭刻

BRC20 / ORC20

- 利用 Ordinals 协议的一部分实现的可分割的代币协议
- 仅仅使用了 Ordinals 的数据 commit/reveal 方式
- 余额统计采用的是账户模型
- 转移采用了聪铭刻方案
- 每次转账都需要先铭刻、再 reveal、再转移铭刻出来的 utxo
- 收款方收到 utxo 后，该 utxo 就无用了
- ORC20 的改进
  - 兼容 BRC20
  - 用 id 取代名称作为唯一标识符，名称允许超过4字母
  - 增加 nonce 作为交易标识符，允许取消交易

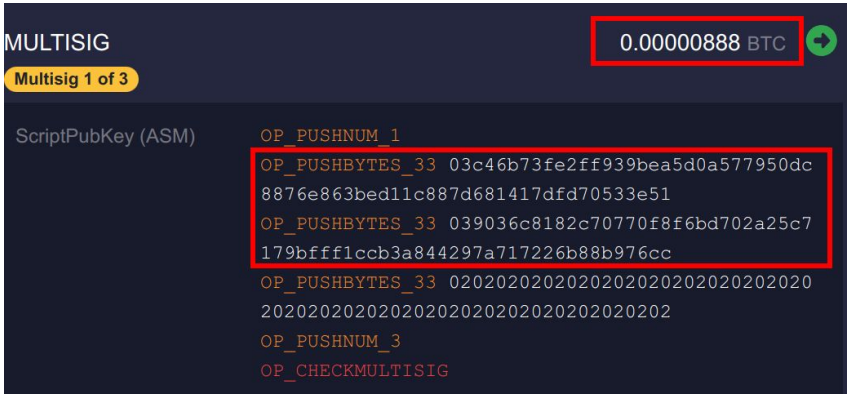
```
{
  "p": "brc-20",
  "op": "deploy",
  "tick": "ordi",
  "max": "21000000",
  "lim": "1000"
}
```

Atomicals / ARC20

- 仍然采用P2TR taproot的 commit/reveal 两步操作写入数据
- 一旦铸造成功，即可随意转移
- 相对于 Ordinals，自定义了更多的应用，包括 ARC20，Bitwork 挖矿，名称系统，递归引用等
- ARC20：一币一聪，无须两步操作
- 未来会有 AVM —— 详情未知（猜测是面向索引器编程）

SRC20

- 和 BRC20 的 indexer 逻辑一致
- 支持 1~5 位 tick
- 数据写在多签字段
- 创建了永远无法花费的高价 UTXO
- 不需要 commit/reveal 操作




```
// 部署
{
  "p": "src-20",
  "op": "deploy",
  "tick": "STAMP",
  "max": "100000",
  "lim": "100",
  "dec": "18"
}

// 铸造
{
  "p": "src-20",
  "op": "mint",
  "tick": "STAMP",
  "amt": "100"
}
```

Runes

- 名称至少13个字母，每隔四个月，可以减少一个字母
- 铸造：提供多种灵活的铸造模式
- Indexer 为每一个不重名的代币分配一个唯一 ID
- 转账：在 OP_RETURN 中记录分配给每一个 output 的代币数量
- 看起来就是一个新版本的染色币

铭文类发币协议总结

	附加数据位置	两步操作	余额模型	玩法
BRC20/ORC20	P2TR witness	发行 + 转账	账户	无
ARC20	P2TR witness	仅发行	UTXO	PoW，name
SRC20	伪多签输出	不需要	账户	无
Runes	OP_RETURN	仅发行	UTXO	多种发行模式，可预留

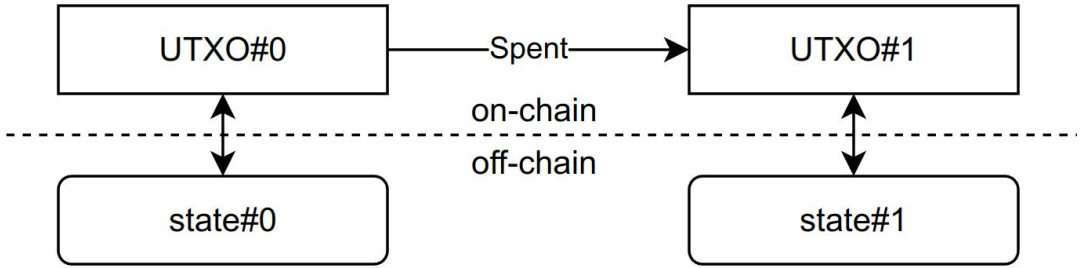
可编程资产协议

铭文类协议的统一问题

- 依赖 indexer，太重，偏中心化
- 没有合约能力，只能卷 meme 赛道
- 大部分资产理论上无法和闪电网络联通
- 所以才有“跨链桥+EVM”范式

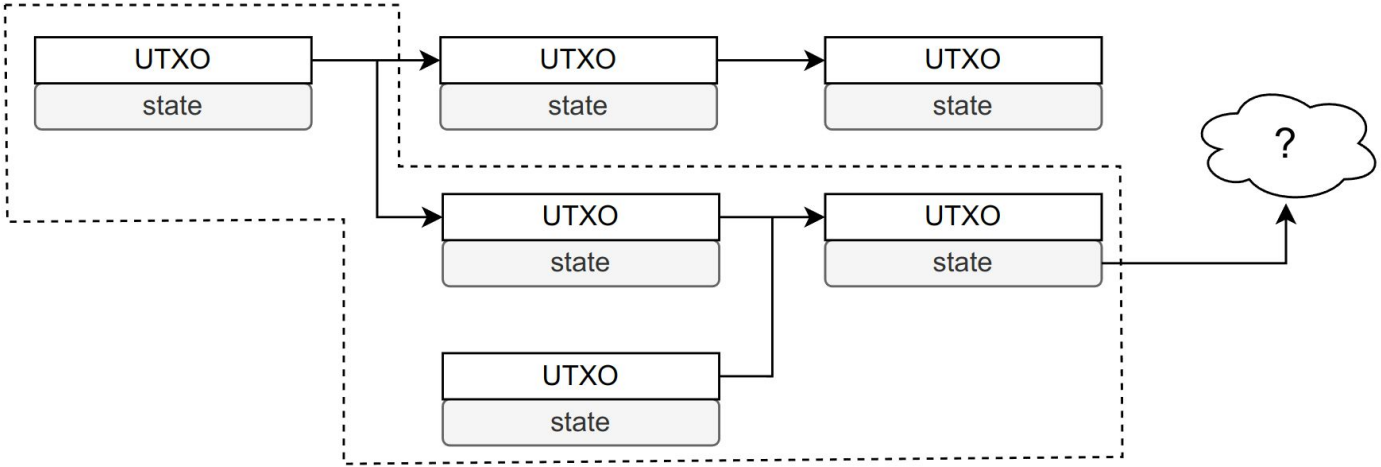
一次性密封（Single-use-seal）

- 最早在 2016 年由 Peter Todd 提出
- 基础思想是一个消息只能被使用（打开）一次
- 基于比特币的一次性密封意味着将所有权绑定在了对 UTXO 的操作上
- 一次性密封和之前基于 utxo 的铭文的区别是，它可以包含**富状态**



客户端验证（Client Side Validation）

- 一次性密封只能解决状态所有权问题，不能解决状态变更的正确性问题
- 客户端验证进一步利用 UTXO 的前后串接特性实现了状态变更的验证，进而为图灵完备智能合约的引入打开了可能性

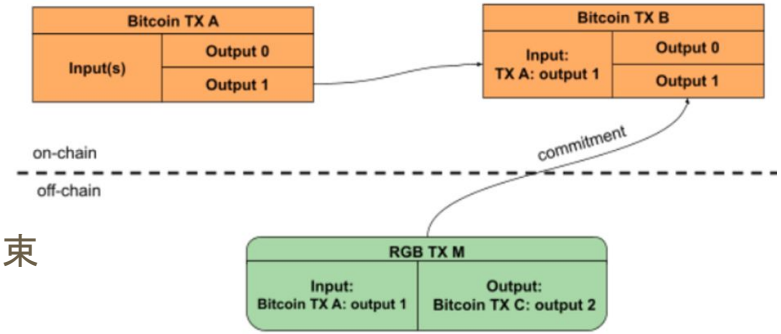


CSV 与 indexer 的区别

- Indexer 需要计算“所有人的所有交易”，CSV 只关心跟自己有关的交易
- Indexer 上实现可编程性要求开发者将代码植入 indexer
- CSV 只需要将代码提供给验证方
- CSV 需要实现大量的链外业务
- CSV 基于比特币轻节点 SPV 即可，Indexer 必须基于比特币全节点

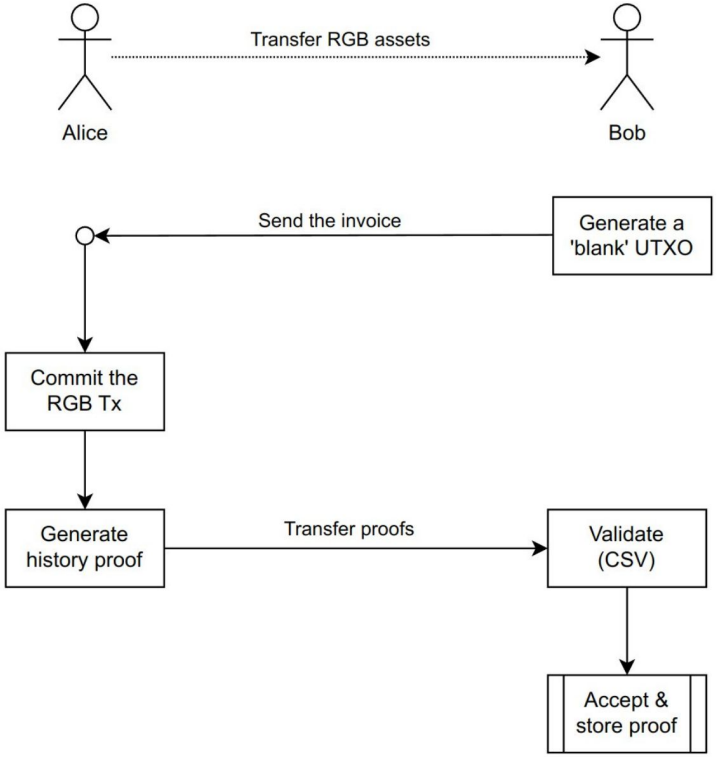
RGB / Taro

- RGB/Taro 为比特币一层带来了图灵完备能力
- 每一笔 RGB TX 都会绑定在一笔 BTC TX 上
  - 通过 OP_RETURN 中保存 RGB TX Hash 来绑定
- 每一个 RGB UTXO 都会绑定在一个 BTC UTXO 上
  - RGB UTXO 的花费条件与 BTC UTXO 相同
  - RGB UTXO 中包含富状态与合约逻辑
- 用户验证 RGB 交易的流程
  - 验证每一笔 RGB TX 绑定的 BTC TX 是否有效
  - 验证每一笔 RGB UTXO 的状态变更是否符合合约约束



RGB 的问题1：交互

- 接收方需要生成一个空的 UTXO 用于接受绑定的 RGB 资产
- 发送方构造交易后，需要生成交易证明，并将历史交易一起发送给接收方
- 接收方需要本地验证



RGB 的问题2：数据安全

- 一方面 RGB TX 的原文数据不上链，带来了很好的隐私保护功能
- 但同时要求用户保存与自己有关的 RGB TX 历史记录，一旦丢失则等价于资产永久锁定
- 交易双方传递交易证明需要专用的 P2P 网络

RGB 的问题3：应用可组合性

- 由于数据的本地化特征，没有任何应用可以拥有全量数据
- 这就导致连不同的钱包之间都不容易做资产互转
- DEX 等产品也很难设计实现

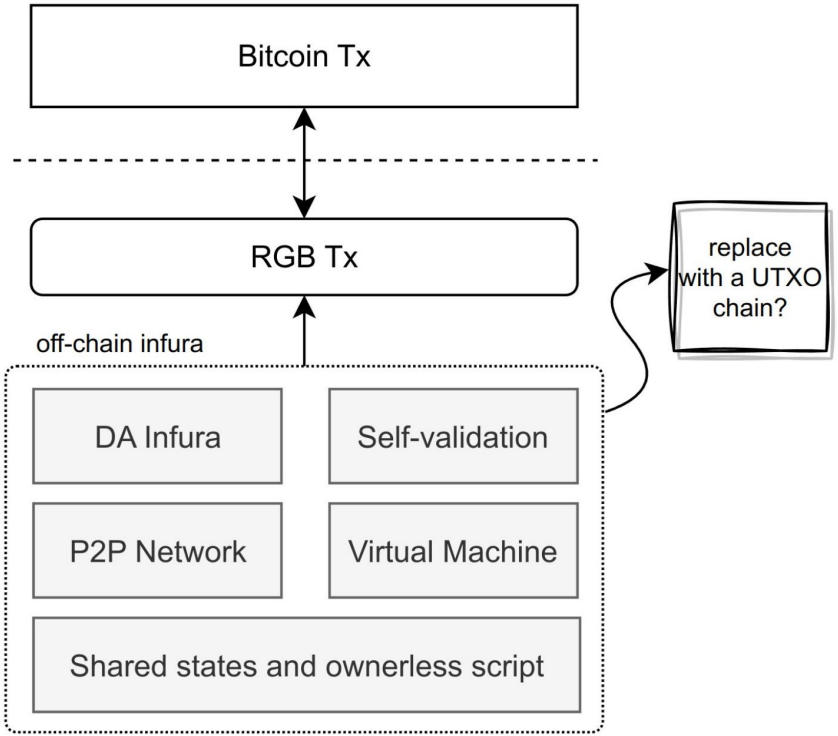
RGB 的问题4：合约与虚拟机

- 虚拟机进度缓慢
- UTXO 本身设计合约就很难
- 同时牵扯到多方的智能合约与 CSV 之间存在某种冲突

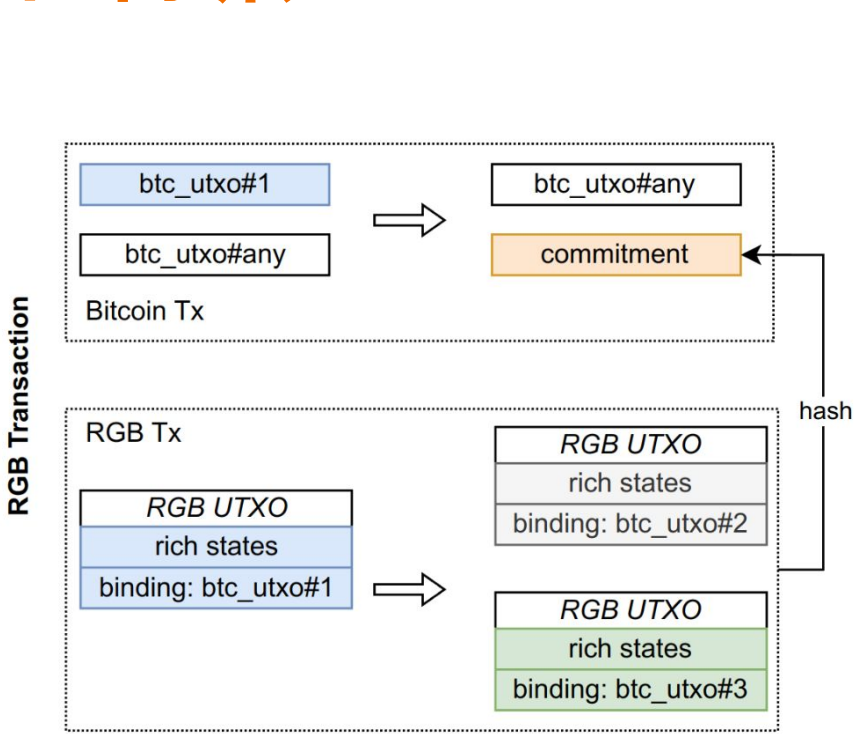
RGB++：比特币 L1 的可编程层

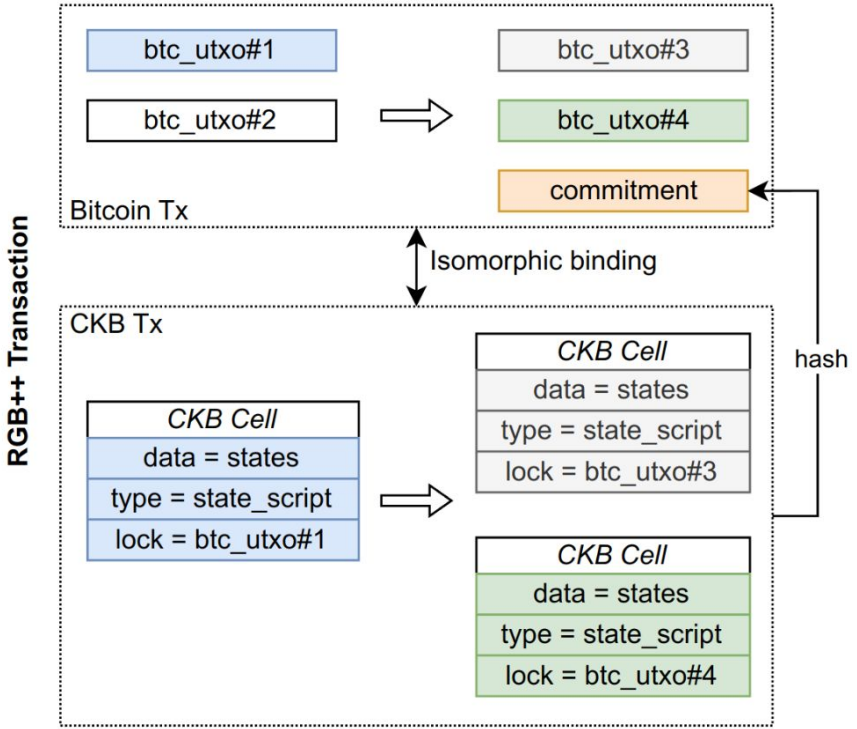
RGB++ 想法来源

- RGB 链外的工作包括 DA、P2P 网络、虚拟机、交易验证等，每一项都比较复杂，需要大量工作
- 这些工作看起来就是一条公链的工作，我们为什么不把他们用一条 UTXO 同构公链代替呢？



同构绑定


RGB Transaction
btc_utxo#1
btc_utxo#any
Bitcoin Tx
btc_utxo#any
commitment
hash
RGB Tx
RGB UTXO
rich states
binding: btc_utxo#1
RGB UTXO
rich states
binding: btc_utxo#2
RGB UTXO
rich states
binding: btc_utxo#3

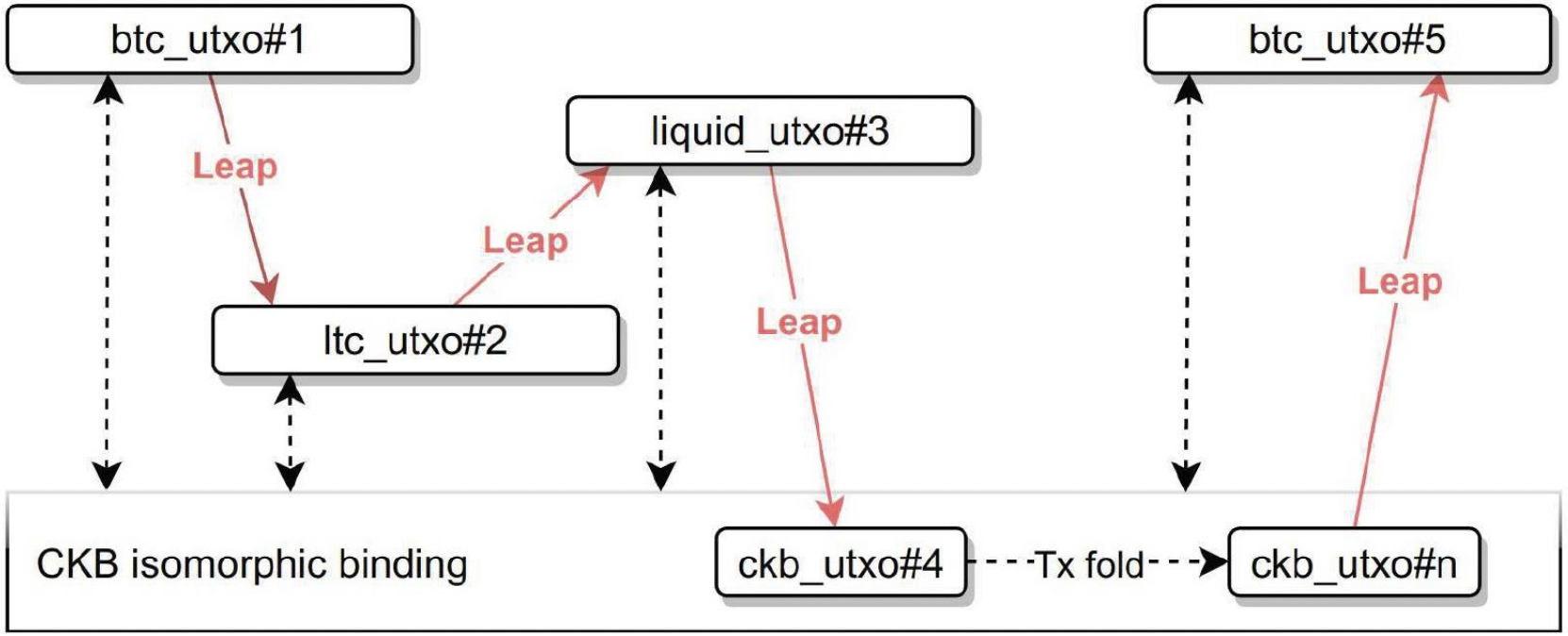

RGB++ Transaction
btc_utxo#1
btc_utxo#2
btc_utxo#3
btc_utxo#4
commitment
Bitcoin Tx
Isomorphic binding
CKB Tx
CKB Cell
data = states
type = state_script
lock = btc_utxo#1
CKB Cell
data = states
type = state_script
lock = btc_utxo#3
CKB Cell
data = states
type = state_script
lock = btc_utxo#4
hash

RGB++ 与 RGB 的区别与改进

- RGB++ 的所有比特币链外数据保存在同构链上
- RGB++ 的交易执行和验证在同构链上
- RGB++ 的操作是非交互式的，同时降低了隐秘性
- RGB++ 可以复用同构链上的所有资产合约、dapp 等基础设施
- 二者都使用比特币地址、比特币钱包，并使用比特币作为网络费用

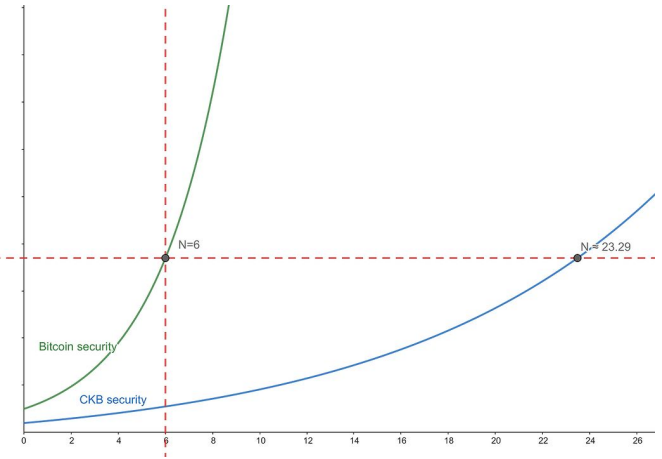
RGB++ 深入讨论

Leap：无需跨链桥的跨链



PoW 带来的跨链安全性

- 不同链之间的 Leap 的安全性来自源链交易的不可撤销性
- Bitcoin 6 个区块确认即可被认为无法撤销（reorg 双花攻击）
- 目前 L1 => L2(CKB) 设置为6个区块确认才可以操作 Leap 过来的资产
- L2(CKB) => L1 呢？
  - PoW reorg 难度与确认数是指数关系
  - 假定相同比例的敌手，CKB 需要 24 个区块确认即可达到比特币 6 个区块确认的不可逆转难度



如何在 RGB++ 上构建智能合约？

- RGB++ UTXO 即使用 Bitcoin UTXO 作为解锁条件的 CKB Cell
- 因此 RGB++ 的智能合约就是 CKB 上的 type（资产）合约
- CKB 上原有的资产合约，包括 Spore，xUDT，mNFT 等都可以直接用在 RGB++ 中
- 使用 Rust/C 即可开发原生的 RGB++ 合约，也可以使用 Js 或 Lua 运行在解释器中快速开发合约

可扩展性

- RGB 原始协议仅解决 Bitcoin 的可编程性问题，并不能解决可扩展性问题
  - RGB 提出了 Prime layer 用来解决这个问题，不过还非常早期
- 闪电网络没有性能天花板，但更适合普通的转账交易，并且在 Bitcoin 上构建通用状态通道网络非常困难
- 过去 Bitcoin 的可扩展性往往集中在跨链桥+EVM侧链上，Bitcoin/UTXO 世界应该有自己的扩展方案
- RGB++ 协议基于 UTXO Leap 可以实现无跨链桥的跨链体验，天然适合进行多链扩展

UTXO Stack

- 一键发链服务，为运营方低成本创建 Bitcoin 同构 Layer2
- 在 CKB 上质押 BTC/CKB，为子链提供安全性
- 与 Bitcoin 之间进行 Leap 跨链
- 资产协议采用 RGB++，可以在子链间，以及 CKB 和 BTC 间任意跳转
- 复用 CKB 智能合约栈
- 复用 Bitcoin 钱包

Thanks

cipher@cell.studio