手册

具有隐私性的 Layer-2 区块链拓展方案

Layer 2 研究团队

HPB 底层技术团队

2022 年 1 月 14 日

目录

1 区块链 Layer 2 扩展方案 **2**

2 具有隐私性的 Layer 2 拓展方案 **2**

3 区块链 Layer-2 设计框架 **4**

3.1 用户交易流程 4

4 算术电路与 Qudratic Arithmetic Program （QAP） **6**

5 算法综述 **7**

5.1 Pre-computing 7

5.2 交易托管计算过程 7

6 Demo **8**

7 总结 **9**

参考文献 **10**

表格

表 1 不同 Layer-2 拓展方案比较 [1] 3

插图

图 1 Layer 2 系统流程图 3

图 2 用户注册时与交易托管的交互 4

图 3 用户交易时与交易托管的交互 5

图 4 用户余额部分内积的 Pedersen 承诺 8

图 5 验证 Pedersen vector commitments 内积算法 9

图 6 验证 Pedersen 承诺内积协议 9

图 7 Experimental results 10

1. 区块链 Layer 2 扩展方案

区块链 Layer 2 [8] 是指构建在现有区块链系统之上的辅助框架或协议。这些协议的主要目标是解决加密货币网络面临的交易速度低和扩展能力差的问题。例如，比特币和以太坊仍然无法每秒处理数千笔交易（TPS），这不利于它们的未来发展。实现吞吐量的提升可以让这些网络得到更有效而广泛的推广使用。在这种情况下，术语“Layer 2”是指针对区块链可扩展性问题提出的多种解决方案, 其中两个主要示例是比特币闪电网络和以太坊 Plasma。

从更广泛的意义上讲，Layer 2 协议创建了一个辅助框架，其中区块链交易可以独立于第 1 层（主链）进行。出于这个原因，这些技术也可以称为“链下”扩展解决方案。例如，闪电网络基于状态通道 (state channel)，状态通道基本上是在线下执行区块链操作，然后将它们报告给主链。另一方面，Plasma 框架由侧链（side chain）组成，侧链本质上是排列成树状结构的小型区块链。

使用链下解决方案的主要优势之一是，主链不需要经过任何结构变化，因为第二层是作为额外的层添加的。主链自身的安全性与 Layer 2 相互独立。因此，Layer 2 解决方案有可能在不牺牲网络安全性的情况下实现高吞吐量。换句话说，由主链执行的大部分工作都可以转移到第二层。因此，主链提供安全性，第二层提供高吞吐量，每秒能够执行数百甚至数千个交易。

目前市场上存在的 layer 2 设计方案，有：zk-Rollup [2]，Plasma [7], Nocust [3] 它们都引入了一个交易托管来线下处理用户的交易。这些方案致力于将大量的计算从主链到 layer 2，但大多数方案没有考虑到用户的隐私性。用户的余额和交易数据在这些系统中都是透明的，这一点很不利于用户的隐私保护。然而，想要同时具有扩展性和隐私性很难，因为增加系统的隐私性会增加更多的数据和计算量，从而降低区块链的吞吐量，增加交易费用。扩展性和隐私性，是硬币的两面，需要适当的取舍。表 1 从安全性、应用性、隐私性与功能评测四个角度对三种主流的 Layer 2 拓展方案以及我们提出的方案进行对比。

我们的项目致力于在隐私性和拓展性的取舍中找到一个平衡，从而为当前市场上的拓展方案增加隐私性，又不会完全降低它们的应用价值。我们像其它拓展方案一样在系统里引入了一个交易托管机构，交易托管将交易打包并给出计算的零证明，将压缩后的交易信息和证明发到区块链上。在我们当前的设计中，交易托管是半可信的：由于交易托管会为自身计算的正确性提供用于验证的证明，因此用户不需要相信交易托管会正确地计算交易，但用户需要相信交易托管会将用户的交易信息保密。

2. 具有隐私性的 Layer 2 拓展方案

和其他 Layer 2 拓展方案相同，我们也在系统中引入一个“半可信”的交易托管，交易托管根据交易更新用户的隐私余额，更新计算是无需信任的，也就是说，交易托管不能随意更改用户数据，根据交易数据计算得到的用户余额数据总是正确的，这个正确性由零知识证明技术保证。但用户需要假定交易托管不会泄露数据，因为交易托管掌握所有的交易信息和用户余额。图 1 示意了系统流程。系统的整体流程如下：

表 1. 不同 Layer-2 拓展方案比较 [1]

方案		Plasma	ZK-Rollup	State Channel [6]	Our Scheme
安全性	No Mass Exit Assumption	✗	✓	✓	✓
	No Watch Tower Assumption	✗	✓	✗	✓
	Cryptographic Primitives	Standard	New	Standard	Standard
应用性	Withdraw Time	1 week	10 minutes	1 confirmation	1 confirmation
	Transaction Finalization Time	1 confirmation	1 confirmation	Instant	1 confirmation
	User Verification	✗	✗	✓	✓
隐私性	Confidential Transaction	✗	✗	✗	✓
	Confidential User Address	✗	✗	✗	✗
	Anonymity Set	Small	Large	Small	Medium
功能评测	Throughput	1k–9k TPS	2k TPS	1–∞ TPS[a]	-[b]
	Cost of Transaction	Very low	Very low	Low	-
	No Collateral Required	✓	✓	✗	✓

[a] 依赖具体的实施方案.

[b] 我们的方案的具体功能评测需要由后续测试结果判定.

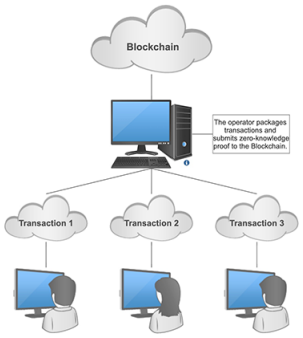

图 1. Layer 2 系统流程图

1. 用户对交易签名后发送给交易托管
2. 交易托管将一段时间内的交易打包，将每个用户更新后的余额做 cryptographic commitment (承诺)，同时计算零知识证明，将承诺和证明发送到底层区块链。
3. 承诺和零知识证明在线上被智能合约验证，验证通过后，与之相关的交易完成。

相对于其他 Layer 2 拓展方案，我们的方案增加了用户的隐私性，因为考虑到完全隐私（匿名交易）需要增加更大的计算负担，我们在隐私性和可行性两方面做权衡后，决定先实现用户交易数据隐私，在此基础上的后续研究将实现匿名交易。

3. 区块链 Layer-2 设计框架

符号说明 设计流程中用到的符号说明如下：

- Op : 交易托管
- SC : 智能合约
- r : 周期标号
- pk_i : 用户 u_i 的公钥
- b_i : 用户 u_i 的余额
- c_i : 用户 u_i 余额的承诺
- t_i : 用户 u_i 的随机值
- c_{t_i} : 用户 u_i 对随机值 t_i 的承诺
- v_{ij} : 用户 u_i 给 u_j 发送交易的数值
- σ : 签名
- B : 智能合约总余额
- Tx'_{ij}:$(pk_i, pk_j, v_{ij}, \mathsf{Null}, r, n, \sigma_{ij})$: 用户发给交易托管交易的明文
- Tx_{ij}:$(pk_i, pk_j, v_{ij}, \mathsf{c_{ij}}, r, n, \sigma_{ij})$: 交易托管回复给用户的交易，其中的交易值已被交易的承诺隐藏
- CTx_{ij} : (pk_j, c_{ij}) 用户 u_i 发给用户 u_j 的交易的简写
- $\mathcal{T}r$: 交易列表（明文）
- $\mathcal{CT}_r$: 交易简写列表 (交易值由承诺隐藏)
- H : 哈希函数
- $\{x_i\}_{i=1}^N$: 数值集合 $\{x_1, ..., x_N\}$, 大括号代表一个数值集合。

3.1 用户交易流程

注册 用户注册时，需先向交易托管发送一个交易需求，包含用户预存额度及用户签名，交易托管返回给用户一个随机值和随机值的 Pedersen 承诺，并对这些信息进行签名。然后用户直接与智能合约交互，预存一定量的余额，并且储存随机值的承诺。图 2 为用户注册时与交易托管的交互过程。

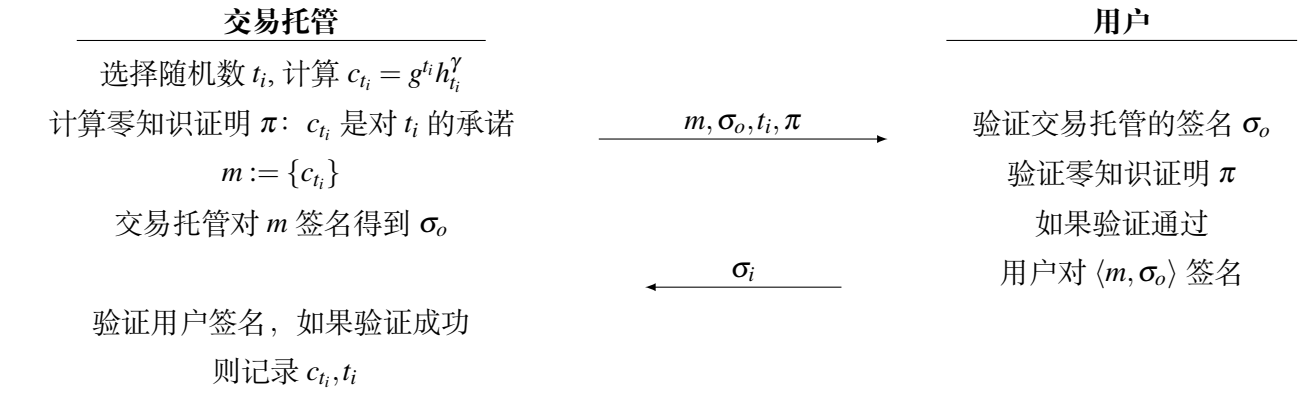

图 2. 用户注册时与交易托管的交互

用户交易 图 3 描述了交易过程中，用户与交易托管的交互过程。用户交易时，将交易和签名发送给交易托管，交易托管验证交易合法性之后，如果这是在一个 epoch 中用户的第一个交易，交易托管将返回给用户以下信息：

1. 一个随机数 t 和其 Pedersen 承诺
2. 对用户剩余额和交易值的 Pedersen 承诺
3. 用户在此 epoch 里完成的所有交易的交易值的 Pedersen 承诺列表。
4. 对以上信息的签名

如果这不是用户在一个 epoch 中发送的第一个交易，交易托管只需要发送 2, 3 和 4.
用户检查合法性，然后返回给交易托管交易值 Pedersen 承诺列表的用户签名。

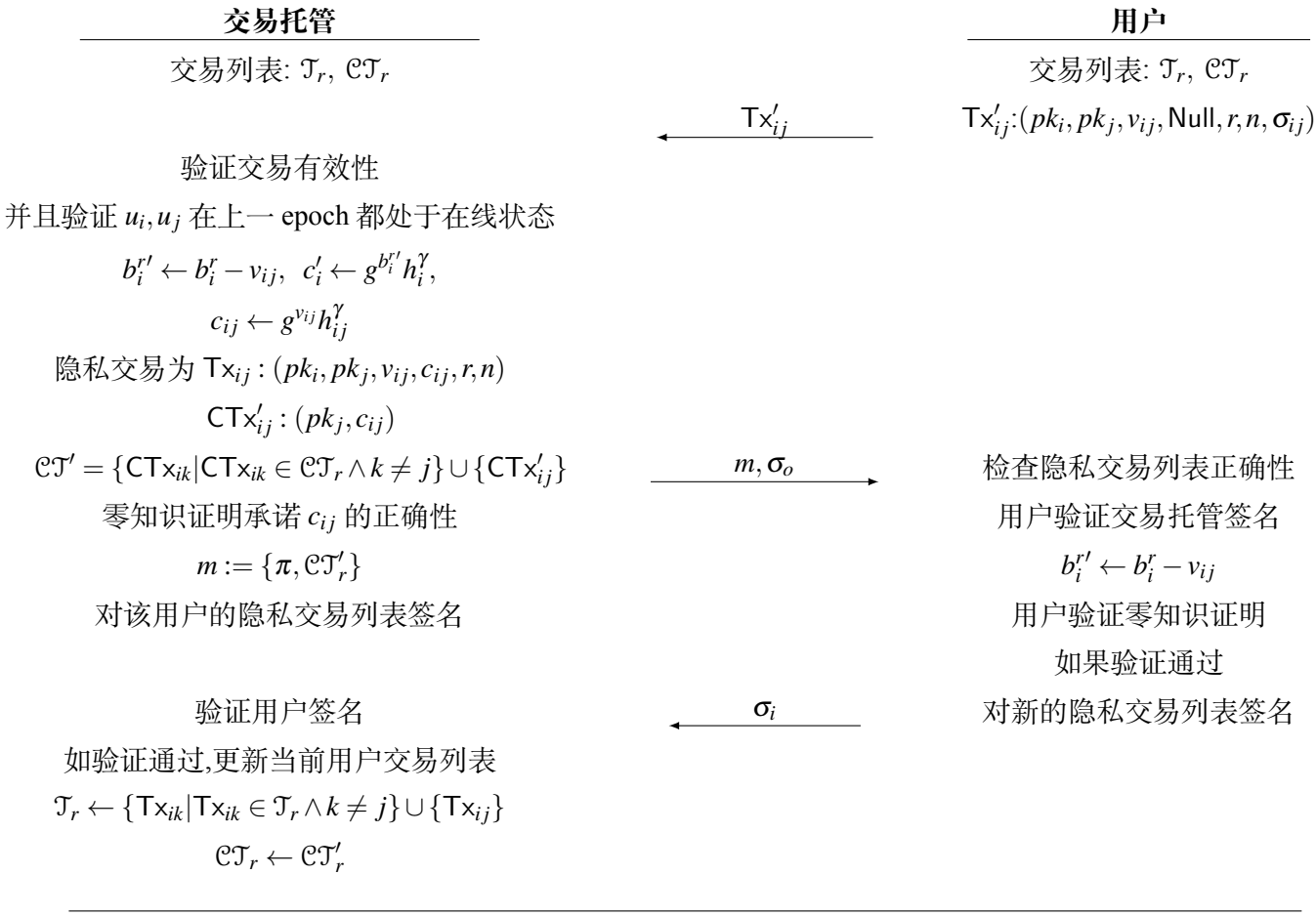

图 3. 用户交易时与交易托管的交互

交易托管链上更新状态 交易托管在 epoch 最后，将收集到这个 epoch 时间段内所有想要发送交易的用户的交易列表以及这些交易的承诺和来自用户的签名。交易托管根据这些交易计算用户的新的余额。最后，交易托管将这些数据发送到智能合约上：

User index	交易列表和用户签名	随机值和随机值签名	余额承诺	d
u_1	$\mathcal{CT}_1^r$, σ_1	c_{t_1}, σ_{t_1}	c_1	d_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
u_i	$\perp$, $\perp$	c_{t_i}, $\perp$	c_i	d_i
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
u_N	$\mathcal{CT}_N^r$, σ_1	c_{t_N}, σ_{t_N}	c_N	d_N
交易托管的签名: σ_o				
零知识证明: π				

其中 d 是用来隐藏用户余额的一个公开数值，其满足 $b = t - d$, b 是用户余额，t 是交易托管和用户共享的随机值。对于每个在这个 epoch 中没有在线与交易托管选取随机值 t_i 的用户 u_i，智能合约更新 u_i 的余额承诺作为他/她在前一个 epoch 的余额承诺。如果在这个 epoch 中用户 u_j 给用户 u_i 发送了交易，则将这些交易返还给 u_j. 具体为: 若用户 u_j 给用户 u_i 发送价值 C_{ji}，那用户 u_j 余额承诺更新为 $c_j = c_j \cdot C_{ji}^{-1}$

用户提现 用户向智能合约发送提现请求，提现数额为 $b = t - d, t$ 为当前与交易托管共享的随机值，d 为公开的数值，用户同时提交交易托管对随机值 t 的签名

4. 算术电路与 Qudratic Arithmetic Program （QAP）

交易托管对用户余额更新的计算由一个算术电路 C 表示, 算术电路进一步由 Quadratic Arithmetic program [5] 表示。

算术电路 算术电路 C 所需要做的计算包括

1. 由交易数据更新用户余额

   $b_i' = b + \sum_{j \neq i} v_{ji} - \sum_{j \neq i} v_{ij}$

   其中 v_{ji} 是当前 epoch 中该用户收到的交易值，v_{ij} 是用户发送的交易值。

2. 用户交易值，用户余额的 range proof，即确保这些数值为正，并且在正确的范围内。

3. 智能合约中存贮的 Token 值与用户行为保持一致。

4. 计算 $b_i = s_i - t_i$，得到 b_i。

QAP 一个算数电路可以表达成一个二次多项式：

$$\sum_{i=1}^{n} a_i u_i(X) \cdot \sum_{i=1}^{n} a_i v_i(X) = \sum_{i=1}^{n} a_i w_i(X) + h(X)z(X)$$

其中 a_i 为电路各个导线的值，其中包括所有用户的余额，交易值，所有随机值等。$u_i(X), v_i(X), w_i(X)$ 为由电路算数结构计算而来的多项式，$h(X), z(X)$ 为由电路连接结构计算而来的多项式，X 为电路结构中门的标号。

算法的思路是，我们将所有用户的余额，交易值，所有随机值等私密数据由 Pedersen 承诺进行隐藏，电路的其他导线值由 Pedersen vector 承诺进行隐藏，最后给出满足以上所有值的算数电路（也就是满足 QAP 方程式），从而证明交易托管进行的计算是正确的。

5. 算法综述

5.1 Pre-computing

1. 规定用户总量 N，由 circuit compiler 计算出关于电路门标号 x 的函数 $u_i(x), v_i(x), w_i(x), h_i(x)$.
2. 每一个 epoch, 智能合约需要产生 (N^2+N) 个新生成元 h_i，用于 Pederson 承诺.

5.2 交易托管计算过程

用户在一个周期内发送交易结束后，交易托管收集到以下数据：

- 每个用户余额的 commitment, 每个用户发给其他用户的价值的 commitment, 这些承诺以 $\{c_i\}_{i=1}^{k}$ 标记。
- QAP 电路中其它的隐私的值的 Pedersen Vector 承诺，以 c_l 表示。
- 电路中公开的值的承诺 $\{c_i\}_{l+1}^{n}$

在有了这些值之后，接下来就可以计算零知识证明，进行 Commit，具体的步骤如下：

1. 将 $h(x)$ 的系数做 Pedersen vector 承诺，得到的结果用 c_h 表示。
2. 将所有的公开的输入值 $((g, h, \{h_i\}_{i=1}^{k}, \{c_i\}_{i=1}^{k}, c_l, c_h, \boldsymbol{g})\{a_i\}_{i=l+1}^{n})$ 做哈希，得到的结果作为 challenge x_1, 算出电路 QAP 多项式的结果 $\{u_i(x_1), v_i(x_1), w_i(x_1), h_i(x_1)\}_{i=1}^{n}$
3. 交易托管选择随机数 t_u, t_v, t_w 并且计算

   $c_u = g^{\sum_{i=1}^{k} a_i u_i(x_1)} h^{t_u}$, $c_v = g^{\sum_{i=1}^{k} a_i v_i(x_1)} h^{t_v}$, $c_w = g^{\sum_{i=1}^{k} a_i w_i(x_1)} h^{t_w}$

   分别以 $\boldsymbol{b} = \{u_i(x_1)\}_{i=1}^{k}, \boldsymbol{b} = \{v_i(x_1)\}_{i=1}^{k}, \boldsymbol{b} = \{w_i(x_1)\}_{i=1}^{k}$ 为参数运行三次图 4 中的运算。

协议输入: $(\{h_i\}_{i=1}^n, \{c_i\}_{i=1}^k, g, h, c \in \mathbb{G}; \{a\}_{i=1}^k, \boldsymbol{b} \in \mathbb{Z}_p^n; \gamma, t \in \mathbb{Z}_p)$　　协议输出: (验证成功/验证失败)

交易托管输入:$(\{h_i\}_{i=1}^n, g, h, \boldsymbol{a}, \boldsymbol{b}, \gamma, t)$

智能合约验证者输入:$(\{h_i\}_{i=1}^n, g, c, \{c_i\}_{i=1}^n)$

(a) 交易托管选择随机数 $\alpha, \beta, t \xleftarrow{\$} \mathbb{Z}_p$, 计算

$$c_0 = h^\gamma, \quad \tau = \prod_{i=1}^n h_i^{b_i}, \quad \Omega = \tau^\gamma h^{-t},$$

$$d_1 = h^\alpha, \quad d_2 = \tau^\alpha h^\beta$$ 并发送 (c_0, Ω, d_1, d_2) 到 $\mathcal{V}$

(b) 计算 $(c_0, \Omega, d_1, d_2 \{h_i\}_{i=1}^n, g, c, \{c_i\}_{i=1}^n, \boldsymbol{b})$ 的哈希值, 作为 challenge x

(c) 交易托管计算 $\theta_1 = \alpha - x\gamma, \theta_2 = \beta + xt$

(d) 智能合约验证者计算 $\tau = \prod_{i=1}^n h_i^{b_i}$, 当且仅当以下条件满足时，验证成功

$$c = \frac{\prod_{i=1}^n c_i^{b_i}}{\Omega} \quad \wedge \quad d_1 = c_0^x h^{\theta_1} \quad \wedge \quad d_2 = \Omega^x \tau^{\theta_1} h^{\theta_2}$$

图 4. 用户余额部分内积的 Pedersen 承诺

4. 交易托管选择随机数 s_u, s_v, s_w 并且计算

$$c_{\boldsymbol{u}} = g^{\sum_{i=k+1}^{l} a_i u_i(x_1)} h^{s_u}, \quad c_{\boldsymbol{v}} = g^{\sum_{i=k+1}^{l} a_i v_i(x_1)} h^{s_v}$$

$$c_{\boldsymbol{w}} = g^{\sum_{i=k+1}^{l} a_i w_i(x_1)} h^{s_w}$$

$$c_{hz} = g^{h(x_1) z(x_1)} h^{s_h}$$

以 c_l 作为以下算法中的 c_a, 分别以 $\{u_i(x_1)\}_{i=k+1}^l, \{v_i(x_1)\}_{i=k+1}^l, \{w_i(x_1)\}_{i=k+1}^l$ 为向量 $\boldsymbol{b}$ 进行以下算法三次, 每次对应的 $c_{\boldsymbol{ab}}$ 分别为 $c_{\boldsymbol{u}}, c_{\boldsymbol{v}}, c_{\boldsymbol{w}}$. 再以 c_h 为作向量 c_a, 向量 b 则为 $\{z(x_1), x_1 z(x_1), x_1^2 z(x_1), ..., x_1^{n-2}\}$, 以 c_{hz} 作为对应的 $c_a b$ 运行图 5 中的算法。

5. 进行 Prove，验证 QAP 方程式是否成立，也就是

(a) 计算

$$c_a = c_u \cdot c_{\boldsymbol{u}} \cdot g^{\sum_{i=l+1}^{n} a_i u_i(x_1)}, c_b = c_v \cdot c_{\boldsymbol{v}} \cdot g^{\sum_{i=l+1}^{n} a_i v_i(x_1)}, c_c = c_w \cdot c_{\boldsymbol{w}} \cdot g^{\sum_{i=l+1}^{n} a_i w_i(x_1)} \cdot c_{hz}$$

(b) 通过图 6 中的算法验证 c_c 承诺的值是 c_a, c_b 承诺值的积。

6. Demo

为了评估 Layer 2 扩展协议的可用性，我们在 *Go* 中给出了一个简单的实现。我们使用 128 位安全的椭圆曲线 secp256k1，余额和交易长度为 8 位，我们测试每个用户的证明大小、交易大小、证明生成时间、验证时间、预处理时间。图 7 展示了我们的实验结果。我们没有在实现上部署优化，因此在时间方面的性能是不可取的，特别是将算术电路问题简化为 QAP 问题的预处理过程需要很长时间，无法在更多用户案例上进行进一步的实验。QAP 更优化的实现方案可以参考 [4] 进行优化，从而快速实现电路参数搭建。实验在 2×24 Xeon 2.4 GHz 内核上进行。

交易托管计算 $(\boldsymbol{g},u,h,c,\boldsymbol{b})$ 哈希值作为 challenge x. 计算 $c=c_{\boldsymbol{a}}c_{\boldsymbol{ab}}^{x}$, $r=r_a+r_{ab}x$, $u=g^x$ 并且以 $(\boldsymbol{g},u,h,\boldsymbol{a},\boldsymbol{b},c,r)$ 为输入运行以下算法：

零知识证明协议:

协议输入: $(\boldsymbol{g}\in\mathbb{G}^n,u,h,c\in\mathbb{G};\boldsymbol{a},\boldsymbol{b}\in\mathbb{Z}_p^n,r\in\mathbb{Z})$ 协议输出: (验证成功/验证失败)

交易托管输入值：$(\boldsymbol{g},u,h,c,\boldsymbol{a},\boldsymbol{b})$

智能合约输入值：$(\boldsymbol{g},u,h,c,\boldsymbol{b})$

当 $n=1$ $(\boldsymbol{a}:=\{a_1\},\boldsymbol{g}:=\{g_1\})$:

(a) 交易托管选择随机数 $\alpha_1,\alpha_2\xleftarrow{\$}\mathbb{Z}_p$, 计算并发送 $d=g_1^{\alpha_1}u^{\alpha_1 b_1}h^{\alpha_2}$ 给智能合约

(b) 交易托管计算 $(\boldsymbol{g},u,h,c,\boldsymbol{b},d)$ 的哈希值作为 challenge x 并发送给智能合约.

(c) 交易托管计算 $\theta_1=\alpha_1-xa_1,\theta_2=\alpha_2-xr$, 发送 θ_1 和 θ_2 给智能合约

(d) 如果 $c^x g_1^{\theta_1}u^{b_1\theta_1}h^{\theta_2}=d$ 成立，智能合约输出验证成功，否则验证失败.

if $n>1$:

(a) 当 $n'=\frac{n}{2}$, 交易托管选择随机数 $r_1\xleftarrow{\$}\mathbb{Z}_p$ 和 $r_2\xleftarrow{\$}\mathbb{Z}_p$. 以以下方式计算 L,R 并且将 L,R 发送给智能合约

$$L=\boldsymbol{g}_{[n':]}^{\boldsymbol{a}_{[:n']}}\cdot u^{\langle \boldsymbol{a}_{[:n']},\boldsymbol{b}_{[n':]}\rangle}\cdot h^{r_1}\in\mathbb{G}$$

$$R=\boldsymbol{g}_{[:n']}^{\boldsymbol{a}_{[n':]}}\cdot u^{\langle \boldsymbol{a}_{[n':]},\boldsymbol{b}_{[:n']}\rangle}\cdot h^{r_2}\in\mathbb{G}$$

(b) 交易托管计算 $\boldsymbol{g},c,\boldsymbol{b},L,R$ 的哈希并将其作为 challenge x

(c) 交易托管和智能合约共同计算：

$$\boldsymbol{g}'=\boldsymbol{g}_{[:n']}^{x^{-1}}\circ\boldsymbol{g}_{[n':]}^{x}\in\mathbb{G}^{n'}$$
$$\boldsymbol{b}'=x^{-1}\boldsymbol{b}_{[:n']}+x\boldsymbol{b}_{[n':]}\in\mathbb{Z}_p^{n'}$$
$$c'=c\cdot L^{x^2}\cdot R^{x^{-2}}\in\mathbb{G}$$

(d) 交易托管计算:

$$\boldsymbol{a}'=x\boldsymbol{a}_{[:n']}+x^{-1}\boldsymbol{a}_{[n':]}$$
$$r=r+x^2r_1+x^{-2}r_2$$

(e) 以 $(\boldsymbol{g}',u,h,c',\boldsymbol{a}',\boldsymbol{b}',r)$ 为输入迭代的运行此零知识证明

图 5. 验证 Pedersen vector commitments 内积算法

协议输入: $(g,h,c,c_a,c_b\in\mathbb{G};a,b,r_a,r_b,t\in\mathbb{Z}_p)$ 协议输出: (智能合约验证成功或验证失败)

$\mathcal{P}$'s 输入:(g,h,a,b,r_a,r_b,t)

$\mathcal{V}$'s 输出:(g,h,c_a,c_b,c)

i. $\mathcal{P}$ 选择随机数 $\alpha,\beta,r_1,r_2,s_0,s_1$ 并计算
$d_1=g^{\alpha}h^{r_1}$, $d_2=g^{\beta}h^{r_2}$, $c_0=g^{\alpha b+\beta a}h^{s_0}$,
$c_1=g^{\alpha\beta}h^{s_1}$. $\mathcal{P}$ 发送 (d_1,d_2,c_0,c_1) 到 $\mathcal{V}$

ii. $\mathcal{P}$ 计算 Challenge 为 $(g,h,c_a,c_b,c,d_1,d_2,c_0,c_1)$ 的哈希值.

iii. $\mathcal{P}$ 计算 $\theta_a=\alpha-ax$, $\theta_b=\beta-bx$,
$\theta_1=r_1-r_ax$, $\theta_2=r_2-r_bx$
$\theta_{ab}=x^2t-xs_0+s_1$ 将 $\theta_a,\theta_b,\theta_1,\theta_2$ 发送给智能合约

iv. 智能合约检查 $c_a^x g^{\theta_a}h^{\theta_1}=d_1$, $c_b^x g^{\theta_b}h^{\theta_2}=d_2$, $g^{\theta_a\theta_b}h^{\theta_{ab}}c_0^x=c^{x^2}c_1$, 如果所有等式成立则验证成功，否则验证失败.

图 6. 验证 Pedersen 承诺内积协议

7. 总结

区块链的扩展性，以及扩展后的隐私性都是区块链应用必要解决的问题，本方案的目标是在区块链拓展方案 Layer 2 的技术基础上实现用户隐私性。为此，我们构建了一个高效的 Commit-and-

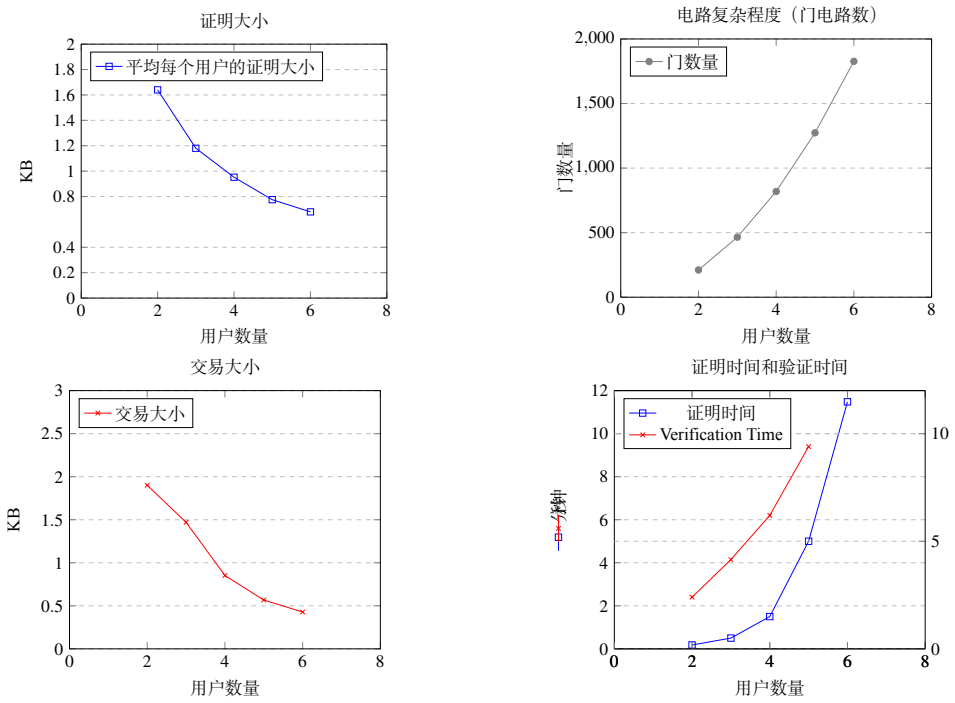

图 7. Experimental results

Prove 零知识证明协议。该方案能在具有扩展性的，支持智能合约的区块链上实现用户交易价值隐私，用户余额隐私。我们对提出的方案进行了测试，测试结果证明方案可以提供隐私属性，但要实现可行性，需要在零知识证明的预处理阶段做优化改进（具体参考 Libsnark 的预处理代码库）。

参考文献

[1] V. Buterin. An incomplete guide to rollups. URL https://vitalik.ca/general/2021/01/05/rollup.html.

[2] A. Gluchowski. Zk rollup: scaling with zero-knowledge proofs. Matter Labs, 2019. URL https://pandax-statics.oss-cn-shenzhen.aliyuncs.com/statics/1221233526992813.pdf.

[3] R. Khalil, A. Zamyatin, G. Felley, P. Moreno-Sanchez, and A. Gervais. Commit-chains: Secure, scalable off-chain payments. Cryptology ePrint Archive, Report 2018/642, 2018. https://ia.cr/2018/642.

[4] S. Lab. libsnark: a c++ library for zksnark proofs. https://github.com/scipr-lab/libsnark.

[5] E. L. Lawler. The quadratic assignment problem. *Management science*, 9(4):586–599, 1963.

[6] A. Miller, I. Bentov, R. Kumaresan, and P. McCorry. Sprites: Payment channels that go faster than lightning. *CoRR abs/1702.05812*, 306, 2017.

[7] J. Poon and V. Buterin. Plasma: Scalable autonomous smart contracts. *White paper*, 2017.

[8] J. Stark. Making sense of ethereum's layer2 scaling solutions: state channels, plasma, and

truebit. *URl: https://medium. com/l4-media/making-sense-of-ethereums-layer-2-scalingsolutions-state-channels-plasma-and-truebit-22cb40dcc2f4*, 2018.