# 基于支付通道的、去信任实时清算协议
# – 使用 Solidity 重新解读闪电网络

OK 区块链工程院

版本：*Draft: 0.3*

日期：*April 2, 2019*

## 摘　要

比特币是一种去中心化的电子现金支付系统，在没有可信金融中介的情况下，通过分布式账本实现了安全可靠的价值传输。优点是通过内在的信任机制，大大降低了对于外部金融监管、风险控制的依赖性。但是同时也产生了三元不可能性问题，既：去中心化、性能、成本三者不可兼得。导致扩容问题长期没有很好的解决方案，虽然比特币已经诞生了十年，区块链技术依然无法在金融领域中大规模应用。

本文分析了比特币扩容困境的根本原因，解释为什么大区块、DAG、分片、侧链跨链、DPoS、PBFT 等扩容方案都无法摆脱三元不可能原理的限制。然后介绍闪电网络的解决方案：基于支付通道的去信任实时清算协议。分析它相对于其它扩容方案的独特性与技术优势。

由于比特币的脚本语言是基于堆栈的逆波兰表达式，使得闪电网络的白皮书晦涩难懂。为了便于读者阅读，本文在保证技术原理不变的前提下，使用 Solidity 语言重新介绍闪电网络的技术原理。然后分析它在性能、费用、实时性等方面的优势，以及技术上存在的限制。另外还介绍了几种相关的技术进展和改进。最后指出，基于支付通道的去信任实时清算协议能够满足现代支付体系的性能需求，并且为金融系统提供了下一代基础设施。

# 目录

## 源码列表

# §1 介绍

闪电网络 (Lightning Network) 是 Joseph Poon 和 Thaddeus Dryja 在 2015 年合著的白皮书中提出的。它在比特币社区中产生了很大反响，在众多关于比特币的论文和白皮书中，被公认为排名第二，其价值仅次于中本聪的创世论文。

由于闪电网络依赖于隔离验证，一直停留在概念和内部开发阶段。从 2017 年比特币隔离验证分叉之后步入正常的发展轨道，2018 年 3 月，Lightning Labs 开发并推出了第一个测试版，之后 ACINQ 和 Blockstream 两家公司也相继推出了不同的实现。根据统计网站 1ml 的数据，闪电网络目前有 7,634 个节点，39,409 个支付通道，支付通道总计有 1,050.19 BTC (约 413 万美金)。说明闪电网络在过去的一年中取得了显著增长。

闪电网络的愿景是解决比特币网络的扩容问题。众所周知，比特币的初衷是实现一个端到端的电子现金系统，为全世界提供一个去信任的、7x24 小时服务的电子支付网络。但是比特币的性能却远远达不到要求。按照平均每个交易 300 字节计算，比特币的平均吞吐量是 5.6 TPS。然而 Visa 的峰值吞吐量可以达到 47,000 TPS。如果对标这个吞吐量，比特币的区块大小要扩张到 8GB 左右，每年要新增 400 TB 的区块数据。这显然是不现实的。

除了闪电网络，区块链社区同时也提出了众多的扩容解决方案，比如大区块、DPoS、DAG、分片、侧链跨链等。这些方案在比特币的分布式账本技术基础上做了优化，例如调整配置参数、优化数据结构、修改共识算法、账本分区处理、优化网络资源管理等等。但是效果都不好，在付出了高昂的代价 (增加存储量、增加网络流量、增加逻辑复杂度、弱化去中心化) 之后，却只是获得了非常有限的性能提升，和 Visa 相比依然还有几个数量级的差距。

唯有闪电网络脑洞大开、另辟蹊径。与比特币不同的是，它使用了基于债权清算的方式完成支付，在比特币网络上构建了二层清算系统。彻底摆脱了分布式账本的“去中心化-成本-性能”的三元悖论约束。不但将系统的并发量上限提升到了几十万 TPS 级别，而且可以做到实时确认，达到了类似于支付宝、微信支付的使用体验。难能可贵的是，它对比特币网络本身的负面影响非常小 (隔离验证对于比特币的负面影响很小)。

闪电网络并没有使用类似于零知识证明那样的高难度技术，但是它的巧妙设计依然令白皮书晦涩难读。市面上也缺乏简单易读而且讲解透彻的科普文章，对于广大金融科技与区块链爱好者和投资者来讲，有很高的学习门槛。所以闪电网络技术的价值长期被误解、被低估。本文使用 Solidity 语言重新实现了闪电网络，绕过了比特币智能合约语言的复杂性，重新梳理了它的基本思想。用通俗易懂的文字，为大家介绍闪电网络的技术原理，总结技术优势和劣势，分析它的适用的场景，最终阐述它在现代电子支付系统中的潜在应用价值。希望能帮助广大读者更深入的认知闪电网络。

此文的章节结构如下：

- 第二节从数字货币两个基本问题切入，介绍比特币的技术思路，分析比特币扩容困境的根本原因。然后对比所有权交割与债权清算两种支付方式，介绍为什么闪电网络能够跳出三元不可能原理的限制，从而大大提升系统的性能。
- 第三节把闪电网络的基本概念和原理抽象出来，提出虚拟银行、共同承诺、支付通道等新概念，并且提出去信任的实时清算协议。

- 第四节分析闪电网络技术的优点和限制。
- 第五节总结去信任实时清算协议对于区块链和金融科技的价值和意义。
- 在附录 A 中，介绍虚拟银行智能合约的 Solidity 源码，详细阐述如何与 RSMC、HTLC 承诺方案配合的技术细节。
- 在附录 B 中，介绍支付通道技术的几种技术进展和优化，更加强大的 Solidity 智能合约语言，闪电网络的技术可以继续拓展。

# §2 分布式账本的扩容问题

本节从数字货币的双花问题出发，分析区块链扩容的难点的根源是什么。然后对比所有权交割与债权清算在这个问题上的差异。从中总结为什么闪电网络能够摆脱三元不可能原理的束缚，从根本上提高支付的吞吐量。从宏观上理解闪电网络的原理，有助于读者理解微观的技术细节。

## §2.1 数字货币的确权问题与双花问题

从货币的历史上看，货币的去实物化是长期存在的发展趋势。从粮食、到黄金、再到纸钞，货币本身的内在价值逐步减小。货币的内涵不依赖于其自身的价值，而依赖于它代表的价值，或者说依赖于人们认为它代表的价值。所以货币的材质、形状、颜色、图案这些因素不是货币的本质，随着货币的发展这些因素都在不断的变化。货币的最终极形态是数字货币，完全去实物化，彻底摆脱外在的实物载体的限制。数字货币的优点很多，包括便于储藏、转移，易分割，同时在防抢劫、防盗窃等方面有更好的安全性。

虽然数字货币有诸多优点，但是长期以来人们一直使用实物货币，数字货币很晚才出现。很重要的原因是由于数字货币有两个特有的问题需要解决：

- **确权问题**：谁是某一笔货币的拥有者？
- **双花问题**：某一笔货币是不是已经被花出去了？

由于实物货币具有不可复制性，这两个问题很容易解决。但是由于数字货币的复制成本几乎为零，解决方案就不是那么简单了。其中的确权问题相对比较容易，可以通过数字签名可以确定所有权。下面重点讨论双花问题。

## §2.2 所有权交割的模式

比特币是一种端到端的电子货币系统，类似于现金支付的过程，它使用的是所有权交割的模式完成支付。如图 1 是一个日常的支付场景，Alice 向 Bob 支付 10 美元。假如这 10 美元是实物钞票，那么 Bob 只需要验证钞票的真伪即可，不需要关心双花的问题。但对于 10 美元的数字货币，Bob 没有简单的办法确认这 10 美元是否已经被 Alice 花费给了其他人。这个问题的困难在于，Alice 可以把数字货币的所有权转给任何人，所以 Bob 需要访问所有可能的接受者，才能确认 Alice 之前没有花费这 10 美元。

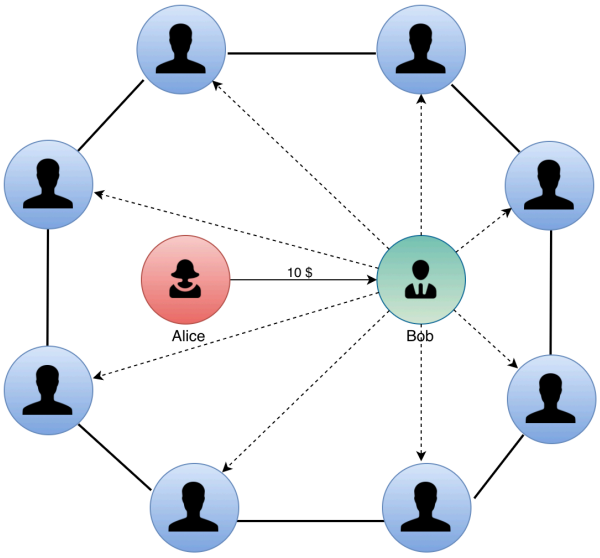


**图 1**

显然穷举并且访问所有人是不可行的。对于这个问题，中本聪在比特币的创世论文中提到：

> ***We need a way for the payee to know that the previous owners did not sign any earlier transactions. For our purposes, the earliest transaction is the one that counts, so we don't care about later attempts to double-spend. The only way to confirm the absence of a transaction is to be aware of all transactions.***

中本聪的解决办法是构建一个分布式账本，包含所有历史交易记录。Bob 根据这个账本直接验证 Alice 是否已经花费了这 10 美元，免去了遍历所有的参与者的麻烦。但是分布式账本需要众多的节点共同维护，这些节点独立管理一份全量副本。为了保证各个节点之间的一致性，同时规避拜占庭故障问题，每一笔交易都要交给所有矿工见证。见证的过程中再附加上工作量证明，令攻击者必须要付出更多工作量才能篡改交易的历史记录。所以每一笔交易的处理成本与节点的个数是正相关的，代价非常高昂。

我们来定性的分析一下。假设参与共识的节点为 $N$，每笔交易消耗的成本 (存储、通信、计算) 复杂度均为 O(N)，分布式账本每秒消耗的成本。那么有公式：

$$\text{每秒消耗的资源}_{\text{存储}} = \text{吞吐量 (TPS)} \times O(N) \tag{1}$$

$$\text{每秒消耗的资源}_{\text{通信}} = \text{吞吐量 (TPS)} \times O(N) \tag{2}$$

$$\text{每秒消耗的资源}_{\text{计算}} = \text{吞吐量 (TPS)} \times O(N) \tag{3}$$

从这个公式可以看到，降低系统成本、提高吞吐量、加强去中心化程度 (提高共识节点个数 $N$)，这三个目标是不可能同时完成的。这是有分布式账本本身的特点决定的，无论怎样设计一个区块链系统，都不可能超出三元不可能性原理的限制。

## §2.3 债权清算的支付模式

回顾上面的分析，我们可以发现，在所有权的交割的模式下，数字货币的支付系统必然会受到三元不可能原理的制约。那么我们换一个思路，看一下债权清算的情况。银行转账就是通过债权清算的方式完成的。其工作原理如图2：

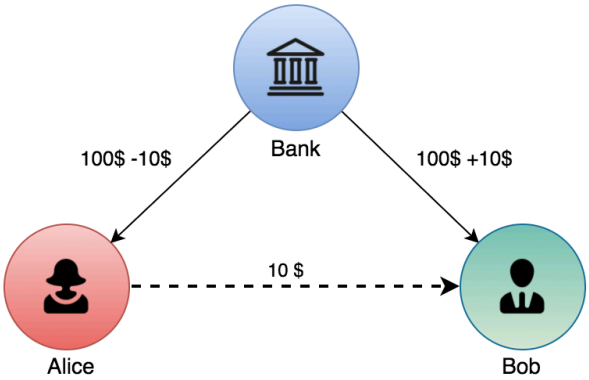


**图 2:** 通过债务清算完成支付

Alice 和 Bob 分别在银行里存入 100 美元，或者说银行分别欠 Alice 和 Bob 100 美元。当 Alice 需要向 Bob 支付 10 美元的时候，银行对 Alice 的债务余额减 10 美元，对 Bob 的债务余额加 10 美元。支付过程中不需要 Alice 和 Bob 之间交割任何实物货币，只需要银行居间调整债务余额就可以了。

因为相对于所有权交割的支付方式，债务清算可以更加高效的解决双花问题。如果 Bob 要确认 Alice 没有双花，只需要让银行与 Alice 共同确认当前最新的余额是多少就可以了。不需要其它任何第三方的参与。相对于比特币的分布式账本来讲，一笔交易的处理成本从 $O(N)$ 降到了 $O(1)$。从根源上摆脱了三元不可能原理的限制，可以非常有效的提高支付系统的性能。

债务清算的支付方式已经在传统的银行机构大规模普及了，而且获得了很大的成功。Visa、支付宝等支付系统背后都有银行、清算中心的支撑。但是因为客户的资产必须长期由银行托管，保持债务关系，银行必须有强大的信用背书和风控制度，防范各种金融风险，保证银行有充足的偿付能力，以免银行被挤兑。为此现代金融系统建立了一整套法律和监管体系，防范这些风险的累积和爆发。然而随着金融体系越来越庞大、越来越复杂，金融监管的成本也越来越高，这些成本最终转化交易的摩擦由消费者买单。

## §2.4 闪电网络的思路

我们对比一下比特币与现代银行系统，二者都是数字货币的支付系统。如果从性能和去信任性这两个维度对比银行系统和区块链系统，我们看到二者恰好是互补的。比特币是基于所有权交割的支付系统，采用分布式账本解决双花问题，所以效率低下，但是它的优点是去信任的，不需要金融中介的参与；反之，银行系统是基于债权清算的支付系统，可以支持高并发、大吞吐量的性能，但是依赖于金融中介的信任，要承担监管和合规的成本。二者像是两个极端，互不相容。

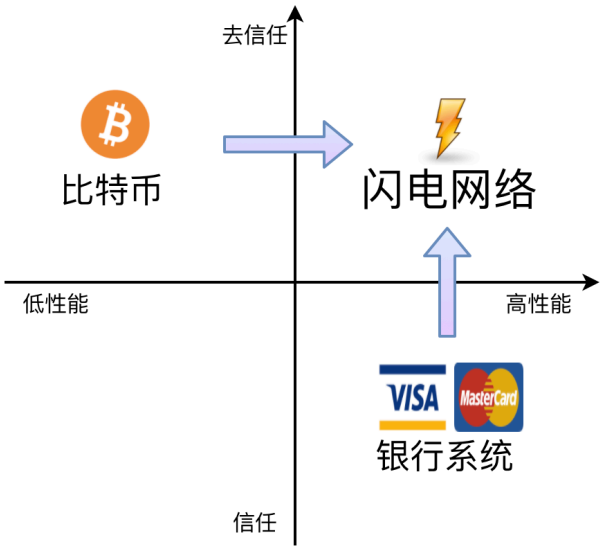


**图 3:** 分布式账本 vs. 银行系统

闪电网络巧妙的整合了二者的优点，一方面利用比特币的智能合约维护债务关系，规避了对于金融机构的依赖；另一方面采用债务清算的方式完成支付，提高了系统的吞吐量。这种新的支付方式我们称之为：基于支付通道的、去信任的、实时清算协议。

# §3 去信任的清算协议

本节先介绍去信任清算的基本概念，帮助读者比较直观的理解闪电网络的思想和理念，想了解更多源代码的读者看完这一节后，请参考附录 A。

## §3.1 基本概念

### §3.1.1 虚拟银行 Virtual Bank

虚拟银行是运行于区块链上的一个智能合约，由支付双方共同协商创建。它模拟一个银行机构作为支付双方的公共债务人。虚拟银行部署之后，按照预先协商的额度，双方向虚拟银行的智能合约注入资金，完成虚拟银行的筹建。如果虚拟银行被清盘结算，资金都返还给支付双方，那么虚拟银行的服务自行终结。

和传统的银行相比，虚拟银行有四个特点：

- **微型**：一个虚拟银行只有两个账户，所以资产负债表也只有 2 条数据。
- **无需信任**：虚拟银行是通过智能合约实现的，继承了智能合约的公开、透明、不可篡改、不可伪造、不可撤销等特点。所以作为公共债务人，虚拟银行没有传统金融机构的风险：比如对手风险、道德风险、流动性风险等。虚拟银行筹建完成之后，它的债务偿付能力永远是 100%，自然也不没有金融监管的成本。提供了无需信任的资产托管服务。

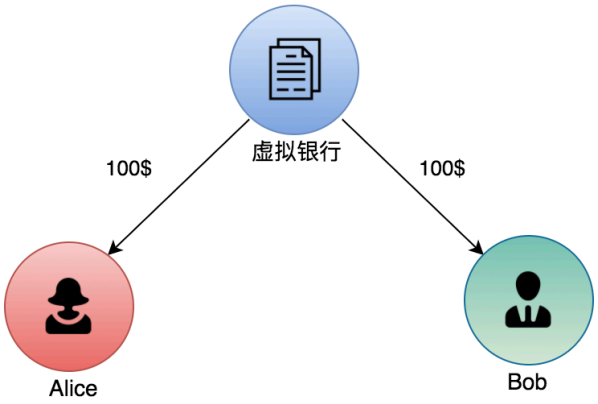


**图 4:** 虚拟银行

- **用户自治**: 虚拟银行只是一个智能合约，并不是一个独立的运营机构。所以银行的资产管理由两个用户共同协商管理。
- **双重签名**：银行只有两个账户，而且总资产不变。一方的资产增加，意味着另一方的资产减少，这是一个零和博弈。为了防止单方面作弊行为，侵害对方的权益，虚拟银行智能合约在处理资产调整请求的时候，要验证双方的签名，保证每一次资产调整都是双方共同的真实意愿。

### §3.1.2 共同承诺 (Mutual Commitment)

每一次微支付，虚拟银行中的资产负债表要做一次调整。双方在链下对债务调整方案达成一致，形成向虚拟银行智能合约的请求消息，并且双方签名。此消息并不立刻广播到链上，而是由双方存储在本地，称之为**共同承诺**。共同承诺是双方真实意愿的表达，是彼此之间对资产分配方案的承诺。共同承诺一旦达成，在任何时候、任何一方都可以将承诺方案广播到链上，虚拟银行都会按照承诺方案结算银行的资产。共同承诺的作用类似于银行支票，虽然没有兑现，但是持有共同承诺就可以无风险的、随时的从虚拟银行中兑现响应的资产。

共同承诺的实现必须满足以下几个要求：

1. **不可伪造**：共同承诺表达虚拟银行双方当事人的真实意愿，任何第三方无法伪造当事人的身份，生成虚假共同承诺。
2. **不可篡改**：对于已经达成的承诺，其中的所有条款无法篡改。虚拟银行智能合约会检查双方的签名，确保承诺的完整性。
3. **可以覆盖**：只要虚拟银行还没有兑现共同承诺，就可以被新的承诺覆盖。对于交易双方来讲，只有最后的一份共同承诺是有效的，之前被覆盖的历史承诺都相当于已经被撤销。共同承诺在技术上有撤销机制。
4. **文义证券**：在共同承诺的条款限定条件下，虚拟银行必须能够随时的、无风险的、按照承诺约定的分配方案结算资产。或者说，共同承诺具有文义证券、无因证券的特点。

通俗的来讲，共同承诺就像是双方共同签署的银行支票，虚拟银行可以在任何时候兑现这张支票。和常规意义的支票不同的：它同时处置两个人的全部资产，一旦兑现某一个承诺，虚拟银行所

有资产被清算，随即关闭。

闪电网络协议中有两种承诺方案：RSMC 承诺与 HTLC 承诺。他们的区别我们后面会讲，但是都满足上述 4 个条件。

### §3.1.3 承诺编号 Sequence

在共同承诺兑现之前，双方可以达成多次共同承诺，撤销旧的承诺，建立新的承诺。这些承诺按照时间顺序编号，以 Sequence 表示。

需要注意的是，闪电网络白皮书中的 Sequence 和本文的定义是不一样的。原文里的 Sequence 是一种时间锁，本文里只是一个简单的编号。但是二者的设计目的都是实现承诺方案的可以撤销机制。所以读者可以暂时忽略二者的技术差异。

### §3.1.4 进攻方 (Attacker) 与防御方 (Defender)

如果一方将共同承诺广播到链上，主动向虚拟银行发起申请, 重新结算资产，此方称之为承诺方案的进攻方。被动的接受对方资产分配方案的一方，称之为防御方。

虚拟银行的资产清算，相当于瓜分银行的总资产，是一种零和博弈。假设双方都是理性的决策者，任何一方都不会做出于对方有利，于己方不利的决策。双方需要一种公平的机制管理共同承诺，规避对手风险，防止对方作弊。

在闪电网络的协议里，一个新的承诺方案先由防御方初审。如果防御方接受此承诺方案，就对此方案签名，然后发送给进攻方进行二审。进攻方持有多个防御方已初审的承诺方案，有权放弃对己方不利的方案。同时有权选择广播共同承诺的时间，当他觉得合适的时候，再加上自己的签名，广播到链上，向虚拟银行请求结算资产。虚拟银行智能合约检验双方的签名，根据共同承诺的条款，公开、透明的结算双方的资产。

### §3.1.5 对偶承诺方案 Dual Commitment

和现实中的支票不同，共同承诺都是一式两份，双方各持有一份。两份承诺编号一致、分配方案一致。但是攻守位置互换。比如说 Alice 持有的那一份，Alice 是进攻方，Bob 是防御方 (Bob 已经初审签名)；反之，Bob 持有的那一份中，Bob 是进攻方，Alice 是防御方 (Alice 已经初审签名)。这两份承诺方案是一对，互为对偶承诺方案，具有同等效力。虚拟银行可以接受任何一份，但是只能接受一份。一份承诺被兑现，另外一份立即作废。

这样设计的好处有两个：一是保持共同承诺的活性，避免单点故障造成的死锁。因为防御方只能被动等待进攻方向虚拟银行发起请求。假如进攻方故障，不能行使进攻方的职责，防御方可以翻转角色，使用对偶承诺方案，以进攻方的身份完成资产的结算。二是保持灵活性和对称性，任何一方都可以随时主动兑现共同承诺，降低对手风险。

### §3.1.6 撤销锁 Revocation Lock

为了标识一个共同承诺方案已经被覆盖，闪电网络协议设计了撤销锁机制。在每一份承诺方案中，进攻方必须要放置一个撤销锁。不同的承诺编号、不同的承诺方案镜像有不同的撤销锁。如果一共有 $N$ 对承诺方案，那么需要有 $2N$ 个不同的撤销锁。

撤销锁由承诺方案的进攻方管理，它实际是一个随机账户地址，对应的私钥由进攻方创建。如果进攻方要撤销某一个承诺方案，他必须公开对应的撤销锁私钥。反过来说，如果防御方从进攻方拿到了一个撤销锁的私钥，那么他可以相信进攻方确实放弃了对应的承诺方案。

一般来讲，从虚拟银行开始，一共有 $N$ 对共同承诺方案，前面的 $(N-1)$ 对承诺方案已经被覆盖。这些历史承诺方案的撤销锁私钥是公开的，每一方都会保留一份对方的**撤销锁私钥列表**。只有最后一对承诺方案是有效的，其撤销私钥还没有公开。

撤销锁的安全机制：当一个承诺方案提交给虚拟银行的时候，防御方可以查看此撤销锁的编号(Sequence)。如果防御方发现此承诺方案是已经被覆盖的，那么从**撤销锁私钥列表**中，找到对应的私钥，并且生成签名作为凭证，向虚拟银行证明对应的承诺方案是已经被覆盖的。虚拟银行将判定进攻方违约，对进攻方处以罚金。这种情况称之为“**破解撤销锁**”。所以如果进攻方是理性的，他就不会冒着撤销锁被破解的风险提交已经覆盖的承诺方案。反之，提交未被覆盖的承诺方案是安全的，因为防御方不知道撤销锁私钥，也就无法破解撤销锁。

当双方创建一对新的承诺方案的时候，需要交换旧承诺方案的失效私钥，表示双方都只承认新的方案，覆盖旧的方案。在承诺方案兑现之前，双方都要妥善保存对方的**撤销锁私钥列表**，防止对方使用对己方不利的承诺提案结算虚拟银行中的资产。

### §3.1.7 诚信保证金 Fidelity Bond

为了保证承诺方案的公平性，承诺方案会设立**诚信保证金**条款，和撤销锁机制配合使用。当虚拟银行兑现一份承诺方案的时候，防御方被动接受进攻方的资产分配方案，所以防御方的资产份额可以优先结算。而进攻方的所有资产作为**诚信保证金**冻结一段时间。目的是防止进攻方提交一份已经被覆盖的承诺方案，侵犯防御方的利益。这个时间段成为**冻结期 Freeze Time**

在**诚信保证金**的冻结期间内，虚拟银行会等待防御方破解该方案的撤销锁。如果破解成功，防御方可以取走所有的诚信保证金作为惩罚，进攻方就会损失所有资产。反之，冻结期满之后，进攻方可以取走诚信保证金。

### §3.1.8 支付通道

支付双方以虚拟银行托管双方的资产，通过共同承诺重新清算双方的存款余额，以达到价值转移的效果，这种支付工具称之为支付通道。虚拟银行筹建并且达成初始共同承诺，标志着支付通道的开启；虚拟银行根据任何一方提交的承诺方案结算双方的资产，就标志着支付通道关闭。

## §3.2 RSMC 承诺方案

### §3.2.1 RSMC 承诺方案的数据结构

在闪电网络中定义了两种承诺方案。第一种称为 RSMC(Recoverable Sequence Maturity Contract) 承诺方案。本文中，一个 RSMC 使用如图 5 的图形化表示，它包含了承诺方案的最基本元素。其中包括承诺编号、撤销锁、诚信保证金、对偶承诺方案等。

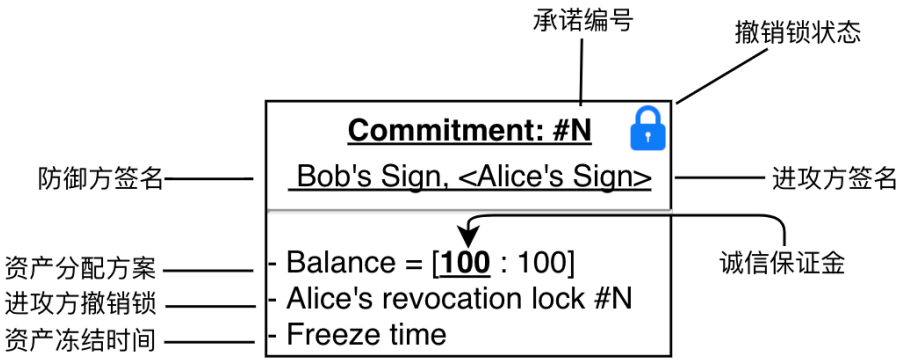


**图 5:** RSMC 承诺方案的数据结构

RSMC 承诺方案分为 Header 与 Body 两部分。

- Header 部分的数据有
  1. **承诺编号**：此例中为 *#N*。如所述，每一个编号的承诺都是一式两份，分别为 Alice 和 Bob 持有。这两份承诺互为对偶承诺，攻守位置互换。左面 Alice 持有的承诺是以 Alice 为进攻方，Bob 为防御方；反之，右面 Bob 持有的承诺以 Bob 为进攻方，Alice 为防御方。
  2. **双方的签名**：防御方作为初审，已经在承诺里面签名。进攻方的签名暂时还空着。比如说左面 Alice 的承诺中有 Bob 的签名"Bob's Sign"，Alice 还未签名，用"<Alice's Sign>" 表示。
  3. **撤销锁状态**: 表示撤销锁对应的私钥已经公开，本承诺方案已经被覆盖。
- Body 部分的数据有
  1. **资产分配方案**：双方约定如何分配虚拟银行的资产，因为这一第一份方案，所以和双方的注资额度是一样的，也是 [100, 100]。
  2. **诚信保证金**: 资产分配方案中，进攻方一方的资产 (下划线标识) 作为诚信保证金将会被锁定一段时间。
  3. **进攻方的撤销锁**: 由进攻方设定的撤销锁，对应的私钥由进攻方管理。如果此方案撤销，进攻方要公开撤销锁的私钥。
  4. **资产冻结时间**：诚信保证金的冻结时间。此时间要足够长，使得防御方有足够的时间审查进攻方提交的承诺方案。

### §3.2.2 通过 RSMC 承诺方案进行支付

假设 Alice 和 Bob 创建了一个虚拟银行，上方初始的资产为 [100, 100]。双方的第一份共同承诺的方案如图 6 所示。这份初始承诺方案按照 [100, 100] 的方式分配银行资产。承诺方案的的编号为 #1，一式两份。左侧的一份以 Alice 为进攻方，右侧的一份以 Bob 为进攻方。这两份互为对偶承诺。

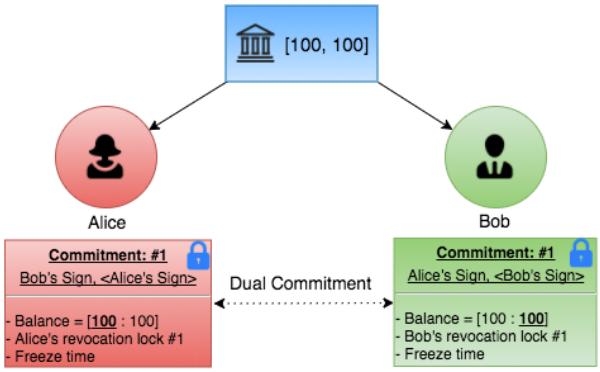


**图 6:** 初始的 RSMC 承诺方案

如果 Alice 向 Bob 支付 10 美元，双方会创建一个新的 RSMC 承诺方案，编号为 #2。分配方案改为：[90, 110]。如下图 7 所示。我了覆盖旧承诺方案，双方互相公开编号为 #1 的承诺方案的撤销锁私钥。同时注意在新的承诺方案里必须使用新的撤销锁。

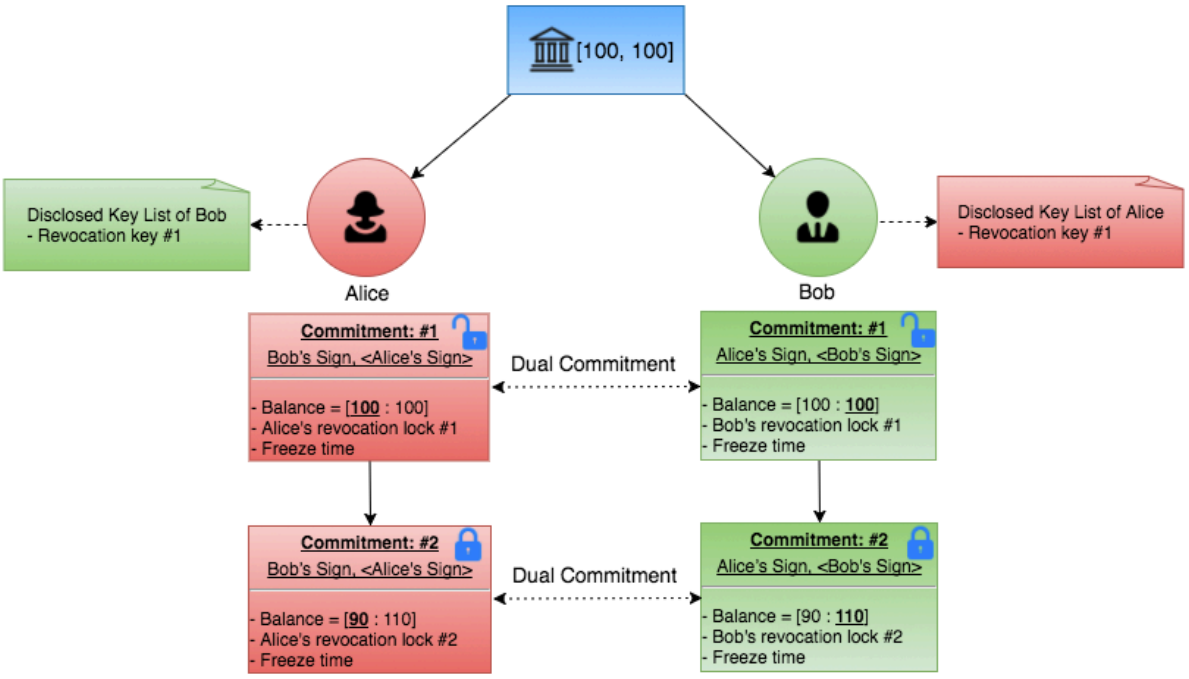


**图 7:** 更新 RSMC 承诺方案

RSMC 承诺方案的特点是没有时间期限，只要没有被撤销，承诺永久有效。任何一方随时可以向虚拟银行提交 RSMC 承诺方案。

### §3.2.3 RSMC 承诺方案的对手风险问题

在图 7 的支付过程中，对于 Alice 来讲，使用新的 RSMC 承诺覆盖旧的承诺方案会损失 10 美元。所以她不会无缘无故的公开编号为 #1 的承诺方案的撤销锁私钥。这种情况下，需要外部的支付环境保证支付的达成。比如说，Alice 向 Bob 购买 10 美元的商品，这时候 Alice 会自愿的撤销旧承诺。

## §3.3 HTLC 承诺方案

### §3.3.1 支付路径

RSMC 协议的局限性在于虚拟银行自有两个账户，只能服务于两个人之间的往来支付。支付双方必须建立直连的支付通道。如果在 $N$ 个人之间建立支付通道，那么每个人需要管理 $(N-1)$ 个支付通道，总计一共有 $(N-1)*N/2$ 个支付通道。闪电网络进一步提出了支付路径的概念，可以将支付通道的复杂度从 $O(N^2)$ 降低到 $O(N)$。

如图 8 所示，Alice 和 Carol 之间没有建立支付通道。但是 Alice 和 Bob，以及 Bob 和 Carol 之间建立了支付通道。这两条支付通道首尾相连，链接成了一个最简单的支付路径。如果 Alice 向 Carol 支付 10 美元，那么可以通过这条支付路径完成。

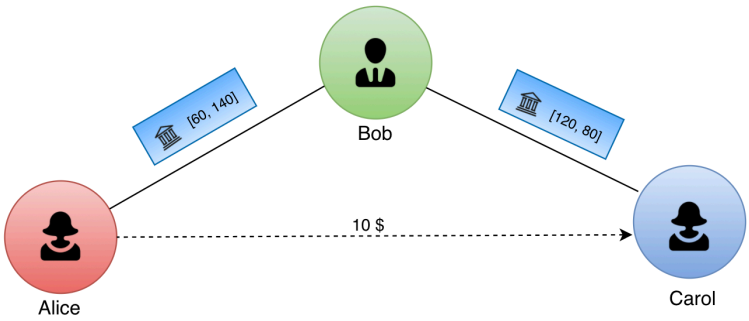


**图 8:** 最简单的支付路径

具体的来讲，就是在 Alice 和 Bob 的支付通道中，将资产分配方式从 [60, 140] 改成 [50, 150]。同时在 Bob 和 Carol 的支付通道中，将资产分配方式从 [120, 80] 改成 [110, 90]。这等价于 Bob 作为中间人帮助 Alice 将 10 美元转到 Carol 手中。

支付路径可以是任意的长度，只要是联通的即可。

### §3.3.2 HTLC 承诺方案的数据结构

在支付路径中，所有的支付通道是互相独立，需要一种机制保证他们的支付具有原子性：要么都完成支付，要么都没有完成支付。为此，闪电网络提出了 HTLC (Hash Time Lock Contract) 承诺方案。和 RSMC 一样，HTLC 中的承诺方案同样也具有不可伪造性、不可篡改性、可覆盖性、文义证券性。

如图 9 所示，这是一个 HTLC 承诺方案的示例。

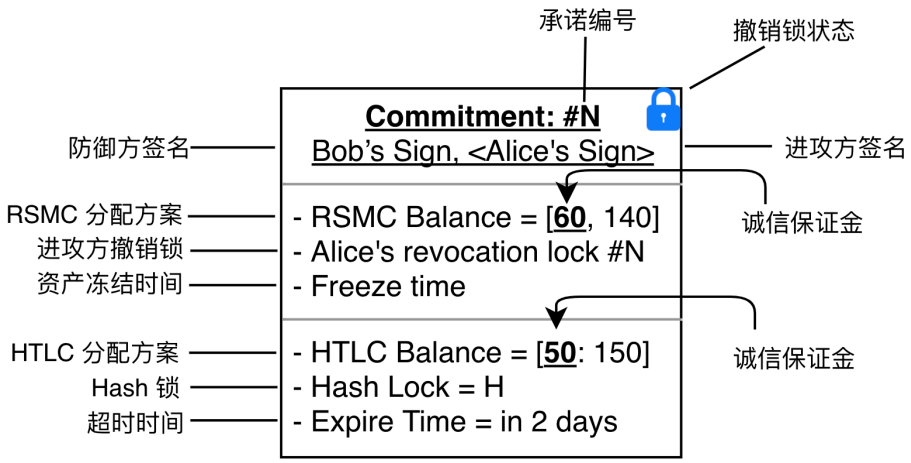


**图 9:** HTLC 承诺方案的数据结构

HTLC 承诺方案也是分为 Header 和 Body 两部分，其中 Header 部分和 RSMC 是一样的。只是在 Body 部分有差异，它在 RSMC 的 Body 部分基础上增加了 HTLC 专有条款:

- HTLC 条款
  1. **HTLC 资产分配方案**: 如果满足 Hash 锁与时间锁两个条件，那么就按照此方案分配资产。
  2. **Hash 锁**：支付的发送方向接收方出示一个 Hash 值 H，接收方必须公开对应的暗语 R，使得 Hash( R ) = H。满足此条件 HTLC 分配方案才有效。
  3. **超时时间**：HTLC 分配方案只能在超时时间内有效。

从语义上讲，HTLC 承诺提供两个不同的资产分配方案。如果满足下面的 Hash 锁和超时时间两个条件，那么 HTLC 分配方案生效。反之，上面的 RSMC 分配方案生效。换句话说，HTLC 部分是一个有条件的、短期有效的分配方案，但是优先级比上面的 RSMC 部分要高。

举个例子，假设根据当前的共同承诺编号为 $\#N$，余额是 [90, 110]。Alice 向 Bob 支付 10 美元。但是前提是 Bob 必须要在 2 小时之内公开暗语 $R$，使得 $H = hash(R)$。那么 HTLC 承诺方案的条款应该是如下图 10 所示：

这一对 HTLC 承诺方案可以解读为：

- 如果虚拟银行在 2 个小时之内接收到此承诺方案，同时提交的暗语 $R$ 满足 $H = hash(R)$，就按照 [80，120] 的方案结算 Alice 和 Bob 的资产。这相当于 Alice 向 Bob 支付了 10 美元。
- 如果虚拟银行再 2 个小时之后接收到此承诺方案，那么按照 [90, 110] 的方案结算 Alice 和 Bob 的资产。这相当于保持原状。
- 其它情况下，不做任何处理。

### §3.3.3 在支付路径上使用 HTLC 进行支付

下面以图 8 为例，介绍在支付路径上如何使用 HTLC 承诺方案完成支付。在支付开始之前，假设 Alice-Bob 的支付通道当前承诺编号为 $N$，如图 11 所示。假设 Bob-Carol 的支付通道当前承诺编

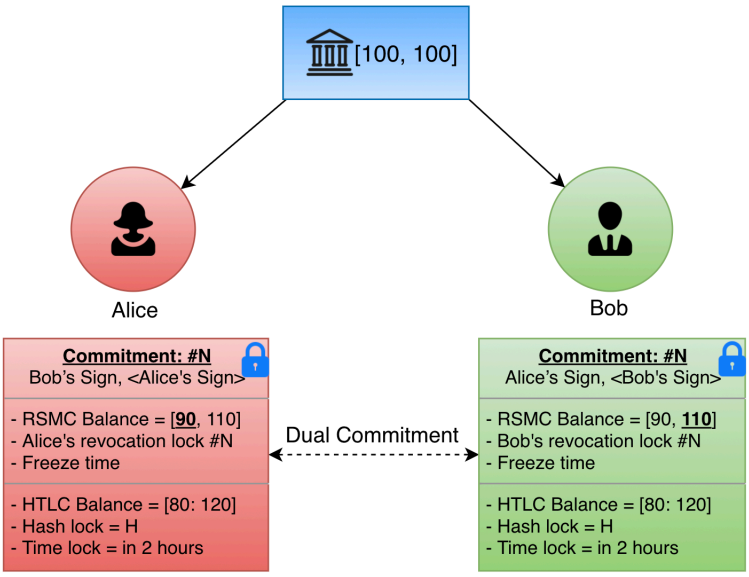


**图 10:** HTLC 承诺方案

号为 $M$，如图 12 所示。

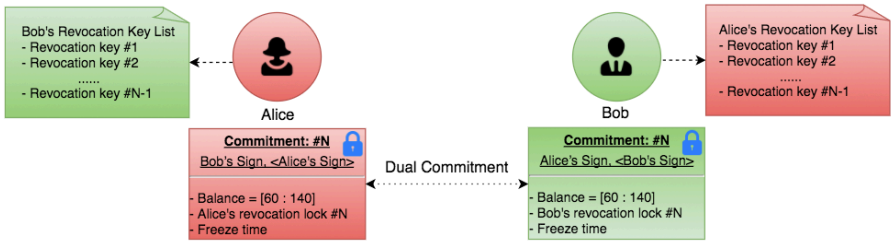


**图 11:** Alice-Bob 的初始状态

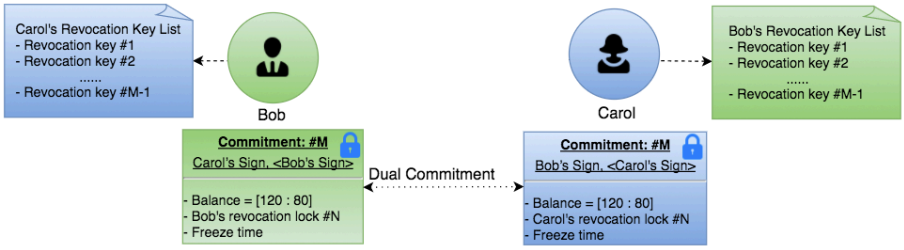


**图 12:** Bob-Carol 的初始状态

整个过程分为五步，如图 13 所示。

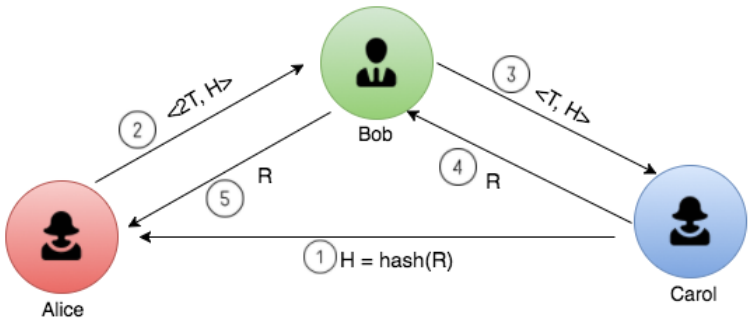


**图 13:** 支付路径

1. Carol 随机产生暗语 R，计算 Hash 值：H = hash(R)，并将 H 发送给 Alice。
2. 如图 14, Alice 和 Bob 达成编号为 #($N+1$) 的 HTLC 承诺：如果 Bob 能在时间 **2 小时**内出示暗语 R，使得 hash(R) = H, 那么 Alice 向 Bob 支付 10 美元。同时覆盖 #($N$) 的承诺方案。

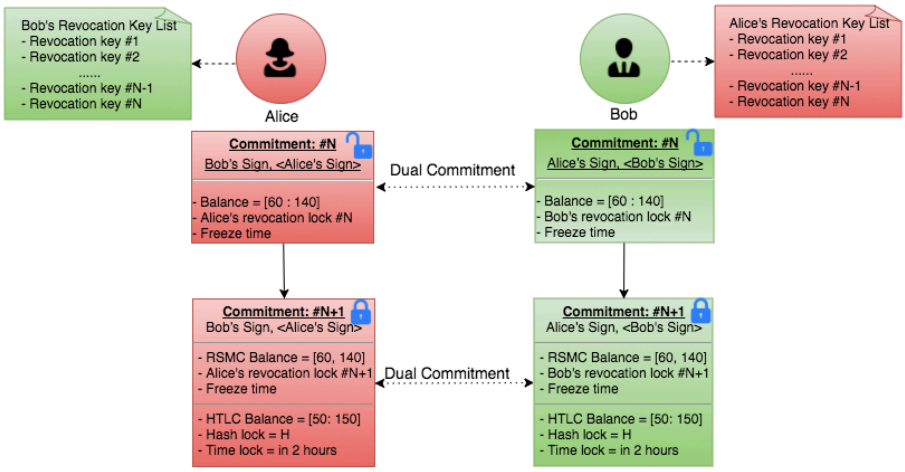


**图 14:** Alice-Bob 签署 HTLC 承诺方案

3. 如图 15，Bob 再和 Carol 达成编号为 #($M+1$) 的 HTLC 承诺：如果 Carol 能在更短的时间 **1 小时**内出示暗语 R，使得 hash(R) = H, 那么 Bob 向 Carol 支付 10 美元。同时覆盖 #($M$) 的承诺方案。
4. 如图 16, 由于 Carol 知道暗语 R 值，在规定的期限 T 内，可以把 R 出示给 Bob，获得 Bob 的 10 美元。之后 Bob 和 Carol 可以再签署一份编号为 RSMC 承诺方案，代替编号为 #($M+1$) 的 HTLC 承诺方案。
5. 如图 17，Bob 从 Carol 那里拿到暗语 R 的时候，和 Alice 之间的 HTLC 承诺还没有过期，向 Alice 出示 R 之后，可以获得 10 美元。然后也可以再重新签署一份编号为 #($N+2$) 的 RSMC 承诺，代替编号为 #($N+1$) 的 HTLC 承诺方案。

最终的结果就是：Alice 通过 Bob 向 Carol 支付了 10 美元，Bob 作为中间人资产并没有变化。HTLC 承诺方案可以在两个联通的支付通道上传递交易，这两个支付通道的交易是具有原子性的。

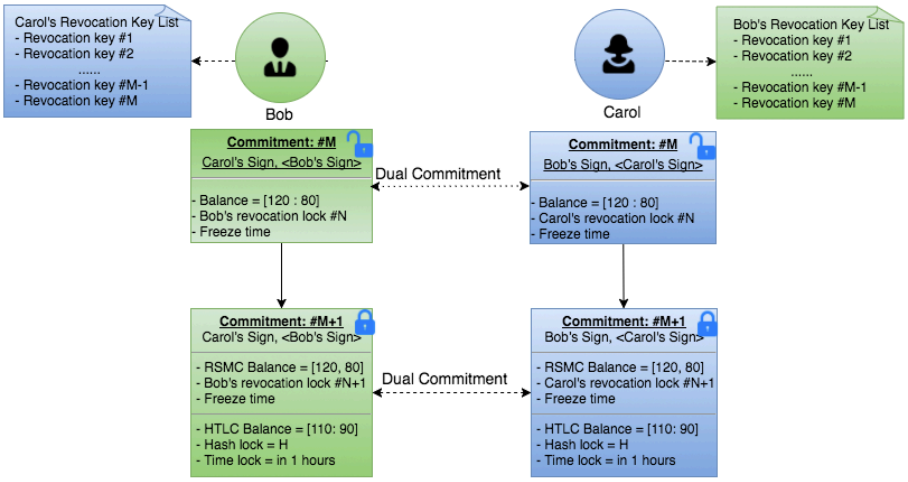


**图 15:** Bob-Carol 签署 HTLC 承诺方案

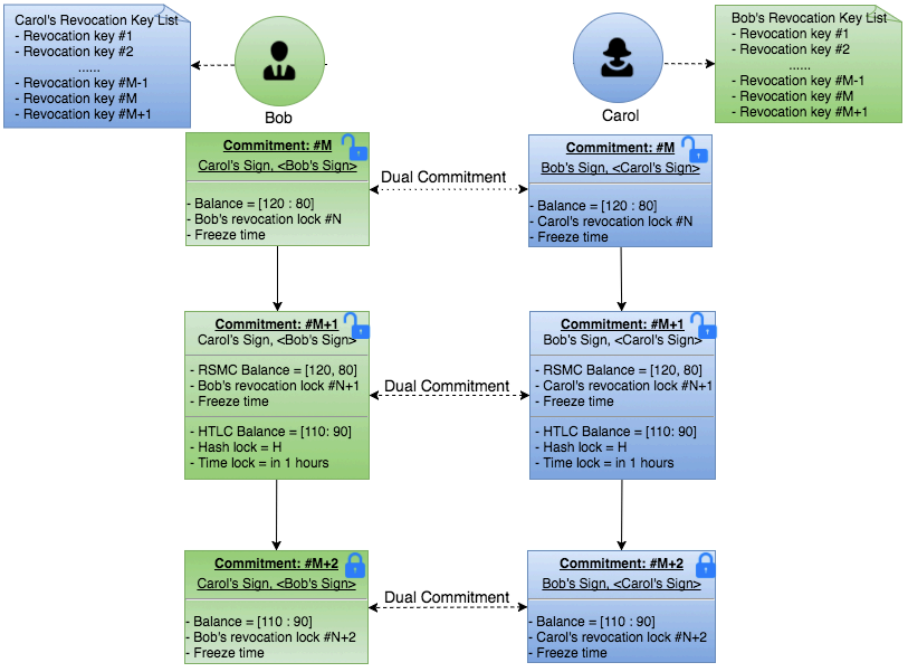


**图 16:** Bob 向 Carol 支付 10 美元

### §3.3.4 支付通道内的对手风险

我们来分析一下，在上一节 §3.3.3 的支付过程中对手风险问题，理性的参与者是否自愿地完成支付过程。我们以 Bob-Carol 的支付过程为例展开分析。

- 首先，支付的接收方无法伪造暗语 R 而欺诈对方，因为在规定的时间内破解 Hash 锁是不可能的。
- 其次，在双方在达成 #$(M+1)$ 号 HTLC 共同承诺之后，要求双方都撤销编号为 #$M$ 的承诺方案。对于接收者 Carol 来讲，新的承诺方案令其有可能得到 10 美元，不会有任何损失，所以会自愿撤销原承诺方案。对于支付方 Bob 方来讲，只有当 Carol 在规定时间内公开暗语 R 的

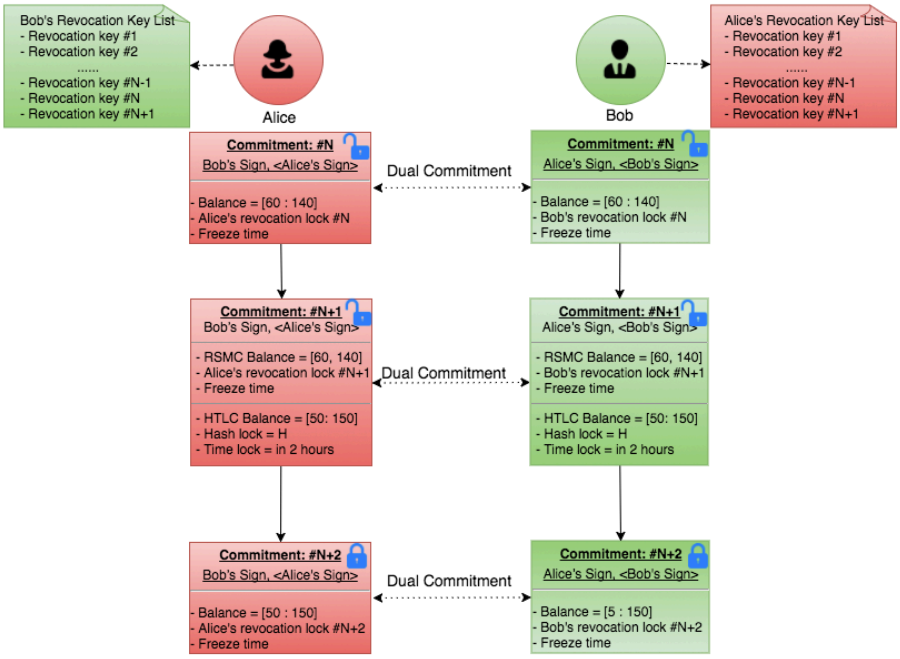


**图 17:** Alice 向 Bob 支付 10 美元

时候才能获得 10 没有, 否则就恢复原状。所以撤销 #$M$ 号承诺也没有什么损失。

- 再次，双方在达成 #$(M+1)$ 号 HTLC 共同承诺之后，要求接受者 Carol 在规定时间内公开暗语 R 。否则超时之后 HTLC 分配方案失效 Carol 就拿不到 10 美元。
- 最后，公开暗语 R 之后，双方可以在一小时的到期之前，建立 #$(M+2)$ 号长期有效的 RSMC 承诺方案。双方要撤销 #$(M+1)$ 号 HTLC 承诺方案。接收方 Carol 无疑会乐意使用新方案代替旧方案。对于发送方 Bob 来讲，不配合也没有意义，Carol 依然能够无风险的获得 10 美元。因为假如 Bob 不配合，想拖延时间。Carol 可以及时提交 #$(M+1)$ 号 HTLC 承诺方案，兑现属于自己的 10 美元。为了保持支付通道，Bob 也必须自愿的更换新的承诺方案。

### §3.3.5 支付通道间的对手风险

下面来分析一下跨支付通道的对手风险问题。注意到 HTLC 承诺方案是沿着支付路径，从发送方向接收方建立；然后再反方向，从接收方向发送方传递暗语 R，依次完成支付。

对于任何一个中间节点来讲，必须先完成右端的支付才能知道暗语 R，否则无法从左侧的支付通道中获得 10 美元的补偿。另一方面，由于左侧的 HTLC 超时时间比右侧的长，当他完成右侧的支付并获得暗语 R 之后，有充足的时间在左侧支付通道获得 10 美元。这就保证了两侧支付的原子性。

对于更长的支付路径，HTLC 承诺方案依然有效。需要注意的是，资金从接收方向发送方传播，每经过一段支付通道，对应的超时时间要增加一个小时，保证资金的发送方有足够的时间向接收到下一个发送方的资金。

进一步推广，所有的支付通道链接在一起构成了**支付通道网络**，其中某一些特定的节点作为中心枢纽，链接其它普通节点。一个节点要向另一个节点支付，只要找到一条联通二者的路径，而且

路径上的每一段都都有充足的额度，就可以通过这条路径完成价值转移到过程。

### §3.4 支付通道的信任机制

综上所述，支付通道的本质是以智能合约托管双方的资产，并且通过双方的自治完成债权的清算。闪电网络设计了一个均衡的二元博弈机制，保证任何一方在自治的过程中不会作弊。这和中本聪的 POW 共识机制有相似的作用。

具体的来讲，在承诺方案的决策过程中，首先是防御方对承诺方案签名，拥有**初审权**，可以拒绝对防御方不利的条款；然后进攻方对承诺方案签名，拥有**复审权**，可以放弃对于进攻方不利的方案。二者的权利是对等的。

在承诺方案的执行过程中，进攻方拥有**主动提交权**，有权选择什么时候提交、提交哪一个承诺方案；防御方拥有**监督权**，有权检查承诺方案的有效性，挑战撤销锁，惩罚进攻方的违约行为；虚拟银行智能合约拥有**执行权**，公开、公平的按照承诺方案的条款处置双方的资产。三者的权利是不同的，既互相独立、又互相制约。

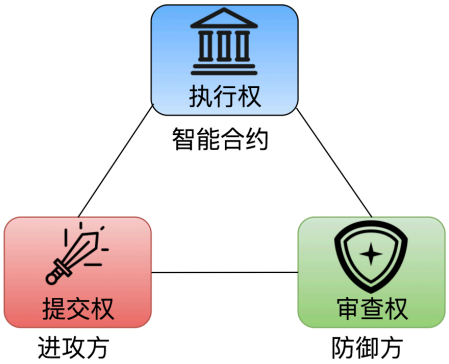


**图 18:** 三权分立

无论是决策阶段、还是执行阶段，虚拟银行的两个用户之间的权利和责任是对等的，谁也无法通过作弊获得不当利益。这种博弈的纳什均衡点，形成一种内生的信任机制，不需要外部的第三方监管与信用保障。就像有“一只看不见的手”，自动促使双方必须诚实的遵守共同承诺。

## §4 技术优势和劣势分析

下面我们逐一分析这种新技术的优势和劣势。

### §4.1 技术优势

根据闪电网络技术的分析，支付通道技术通过链下的共同承诺方案完成支付，把参与共识的范围大大缩小了，会带来很多优势：

- **交易费用低**交易过程中，需要借用中间节点提供的资金流动性，所以需要支付一定的费用。但是由于中间节点是非垄断性的、而且是去信任的，没有监管、合规等成本，费用非常的低。

相对于 Visa 的 2 3% 的费率，闪电网络收取的费用几乎可以忽略不计。

- **交易实时确认**支付的过程中无需大量节点参与共识，只需要参与者之间交换共同承诺方案的参数和签名信息，可以实时的完成交易的确认。具体的来讲，交易确认的时间包含：
  - 支付通道路由选择。如果支付双方没有直接联系的支付通道，需要通过路由算法找到合适的支付路径。
  - 支付通道数据交互。每一次承诺方案编号加一，需要 2 次数据交换：一次是交换新的承诺方案，一次交换旧承诺方案的撤销锁私钥。如果仅仅涉及到 RSMC 承诺方案，每次支付编号只加一次；但是一般情况下是 HTLC 和 RSMC 互相配合使用，承诺方案编号增加 2，所以一共需要 4 次数据交互。

  所以，连个节点之间一次交互的时间一般在几十至几百毫秒之间，所以总的确认时间大约在几百毫秒至几秒之间。
- **高并发性**

  不同的支付路径是互相独立的，可以并行执行，相对于链上的交易，系统的并发性被大幅提升。我们大致的估算一下闪电网络并发性的理论上限。对于正常的一个支付通道来讲，只有建立和关闭发生在链上，一共有 4 个交易 (两个充值、两个取款，闪电网络不需要部署虚拟银行智能合约)。那么底层区块链的吞吐量限制了支付通道的数量，假设平均每个通道的生命周期是 N 天，对应的通道数量的计算公式为：

$$\text{支付通道数量上限} = \text{比特币 TPS} \times 3600 \times 24 \times N \div 4 \tag{4}$$

  根据闪电网络统计网站 1ML的数据，支付通道的平均生命周期 N 为 54.9 天。假设比特币的吞吐量为 3.33 TPS，那么对应的支付通道上限为: 3,952,800。

  根据 6 度空间理论，任何两个陌生人之间的间隔不会超过六个人，也就是说，支付路径的最大长度一般不会超过 6。那么根据下面公式可以算出，闪电网络可以同时支持 658, 800 个支付的并发执行。

$$\text{闪电网络并发量} = \frac{\text{支付通道数量上限}}{\text{平均每笔交易占用通道数}} = \frac{3,952,800}{6} = 658,800 \tag{5}$$

- **数据存储小**

  交易在链下完成的另一个好处是节约数据存储成本。交易数据只需要在参与方之间传播，历史承诺方案可以立刻丢弃，只需要保存对应的撤销锁私钥列表。相对于链上交易来讲，大大降低了数据存储的需求。
- **隐私性**

  除了支付通道的开启和关闭，大部分交易都发生在区块链之外，没有广播到链上，只有通道的参与者了解交易的信息。因此所有微支付几乎都无法追踪。对于支付具有天然的隐私保护作用。

### §4.2 缺点

另一方面，支付通道的支付方式也有限制条件和缺陷，只有在合适的条件下才能充分发挥它的价值。

- **高频往来交易**
  支付通道的建立是有成本的，开启和关闭支付通道需要至少 4 笔交易。如果在支付通道的存续期间只有少量的支付，那么支付通道的意义就不大了，不如直接在链上交易。最适合支付通道的应用场景是双方需要频繁的往来交易，比如说银行间支付清算业务。两个银行之间建立支付通道，为其用户之间的往来支付提供清算服务，可以最大的提高支付通道的利用率。
- **资金锁定**支付通道的中间节点需要提供一定的资产，这部分资产长期处于锁定状态，牺牲了一部分资金的流动性。
- **双方必须在线**虽然承诺方案的协商发生在链下，但是虚拟银行的双方都需要监控链上交易，及时发现对方提交的承诺方案，才能避免自己的损失。所以支付通道的双方依然需要在线监听链上的交易。这个要求对 B 端用户没有太大影响，但是对于 C 端用户来讲，是一个比较强的要求。

## §5 总结，现代支付清算结算系统与应用

在现代金融系统中，有两种并行运作的支付体系：一个是基于所有权交割的现金支付体系，另一个是基于债权清算的银行支付体系。现金支付体系是一种古老的支付模式，具有普适性、门槛低的优势，基本上所有国家和地区都能使用；银行支付体系的优势是：安全、便捷、高效、支持远距离支付；但是需要现代银行系统和金融 IT 系统的支持，对金融监管、风险控制、网络安全、智能终端等基础设施要求很高，一般只有经济较为发达的国家才能建设。这两种支付体系互相独立、互相关联，互相弥补，共同支撑了金融系统的运行。

以比特币为代表的区块链技术基于互联网技术构建了电子现金支付系统，替代了实物现金，大大降低了现金的存储、携带等管理成本，同时也提高了支付的便捷性、安全性。但是由于分布式账本的天然缺陷，交易的成本很高，一直受到扩容问题的困扰，吞吐量低下限制了区块链在金融系统中的普及。

闪电网络采用了债权清算的支付方式，从根本上打开了系统的性能瓶颈。如表 1 所示。虽然债权清算需要双重签名解决确权问题，但是在双花问题上不再需要全局账本，节省了维护分布式账本的成本，大大提升了系统吞吐量。

| | 支付模式 | 确权问题 | 双花问题 | 成本 | 吞吐量 |
|---|---|---|---|---|---|
| 比特币 | 所有权交割 | 一重签名 | 全局账本 | 高 | 低 |
| 闪电网络 | 债权清算 | 二重签名 | 局部账本 | 低 | 高 |

**表 1:** 比特币与闪电网络的对比

同时，它使用虚拟银行智能合约代替传统的银行机构托管用户的资产，在链下构建了均衡的博

弈规则对资产进行清算。这种新的清算方式一方面保留了无需外部信用监管的优势，同时也提升了支付系统的容量和隐私性。

**从某种意义上说，比特币通过现代互联网技术还原了最原始的价值支付方式，而闪电网络又把我们拉回到了现代。** 这两种区块链技术互相关联，互相弥补，共同为金融提供新的支付解决方案。相对于传统的支付系统，区块链技术有以下优势：

- 低摩擦

  区块链有内在的信用机制可以防范支付过程中的对手风险、交易风险，大大节约了对于金融机构的监管、合规成本。这些成本上节约最终会降低支付系统的摩擦，令消费者受益。

- 轻金融

  传统金融机构有体量大、成本高、组织结构复杂等特点，没有为社会所有的人群提供有效的服务。而是倾向于为大机构、大企业、富有群体提供金融服务。相对来讲，使用区块链开展金融业务的机构更加灵活、更加轻量级，能有效、全方位地为社会所有阶层和群体提供服务。

- 易普及

  区块链基于互联网技术，而且对于可信金融机构的依赖程度低。对于欠发达地区来讲，构建电子支付系统的门槛大大降低了。使金融负能够惠及全球更多人口。

在未来几年，区块链作为基础设施逐渐应用于各种金融服务，逐渐表现出强大的技术创新性，市场和消费者肯定受益颇丰。

# 附录 A　技术详解：去信任的清算协议

本节具体阐述闪电网络清算协议的技术细节。对于技术细节不感兴趣的读者可以略过此部分，只需阅读§3。

闪电网络的技术细节晦涩难懂，而且工程实现的复杂度也比较大。部分原因是由于闪电网络的清算协议基于比特币协议，其智能合约是通过堆栈式指令编写，类似于汇编语言的风格。而以太坊的智能合约编程语言 Solidity 的语法接近于 JavaScript，是一种高级编程语言的风格，相对来讲有更好的可读性。所以本文基于 Solidty 重新表达闪电网络协议。在保持技术原理一致性的前提下，尽量提高可读性，帮助读者降低学习的门槛。

## A.1　虚拟银行智能合约

不失一般性，假设 Alice 和 Bob 两个用户在某一段时间内需要频繁的往来支付。于是双方协商建立共同的虚拟银行智能合约。这个合约模拟了一个微型银行，只有 Alice 和 Bob 两个账户。双方约定分别在虚拟银行中存入 100 美元，用 [100, 100] 表示 Alice 和 Bob 在资产负债表的初始余额。

虚拟银行智能合约的源码位于：GitHub: Solidity Bidirection Payment Channel。

### A.1.1　合约数据结构

首先介绍虚拟银行的数据结构，一共分为三部分:

- 虚拟银行状态 State _state。虚拟银行一共有 4 个状态，如下图所示。


**图 19:** 虚拟银行状态变换

  1. Funding: 智能合约初始化完成之后进入 Funding 状态，Alice 和 Bob 根据约定的额度，调用 depoit() 往虚拟银行里转账。Alice 和 Bob 都足额注入 100 美金之后，虚拟银行进入 Running 状态。
  2. Running: 此时，Alice 和 Bob 在链下可以进行微支付交易。他们随时可以提交资产承诺提案 (虚拟银行接受 RSMC、HTLC 两种提案)。虚拟银行将根据承诺提案的条款，立刻结算防御方的资产，冻结进攻方的资产作为诚信保证金。然后进入 Auditing 状态。
  3. Auditing: 在诚信保证金冻结期间，防御方可以检查承诺是否被撤销。如果是，那么他可以取走诚信保证金；否则进攻方可以在冻结期满之后，可以取回诚信保证金。无论哪种情况，虚拟银行都会进入 Closed 状态。
  4. Closed: 虚拟银行的所有资产都已结清，支付通道随之关闭。

- 用户数据 Client[2] _clients。

  由于虚拟银行只有两个账户: Alice 和 Bob。所以此数组的长度永远是 2。每一项保存的用户的地址、应存入的余额，以及是否已经足额存款。

- 承诺数据 Commitment _commitment

  变量 _commitment 记录 Alice 和 Bob 双方已经约定的承诺，包含如何分配虚拟银行的资产，以及承诺编号、进攻方、撤销锁、诚信保证金的冻结时间等。

```
/**
* @title Virtual bank smart contract. Support RSMC and HTLC commitments.
* @dev Simulate the lightning network clearing protocol with Solidity programming language.
*/
contract VirtualBank {
    using SafeMath for uint256;
    using ECRecovery for bytes32;
    string[2] constant NAMES = [string("Alice"), "Bob"];
    struct Client {
        address addr;   // Alice's and Bob's addresses
        uint256 amount;      // amount of each account
        bool    deposited;   // whether each account deposit enough fund
    }
    struct Commitment {
        uint32     sequence;
        uint8      attacker;      // defender = 1 - attacker
        address    revocationLock;
        uint       freezeTime;
        uint       requestTime;
        uint256[2] amounts;        // amount[attacker] is fidelity bond
    }
    // enum for virtual bank state
    enum State { Funding, Running, Auditing, Closed }
    // balance sheets
    Client[2] _clients;
    // state of virtual bank
    State _state;
    // commitment data
    Commitment _commitment;
}
```

**源码 1:** 虚拟银行智能合约数据结构.

### A.1.2 构造函数

创建虚拟银行之前，Alice 和 Bob 需要提前协商相关的配置参数，这些参数包括：

1. Alice 和 Bob 的个人账户地址: address[2] addrs。2. Alice 和 Bob 欲存入的账户余额。

构造函数检查输入参数的合法性，初始化用户数据 _clients, 和承诺方案 _commitment 之后，进入 Funding 状态。

```
/**
 * @notice The contructor of virtual bank smart contract
 * @param addrs  Addresses of Alice and Bob
 * @param amount Balance amount of Alice and Bob
 */
constructor(address[2] addrs, uint256[2] amounts) public
    validAddress(addrs[0])  validAddress(addrs[1]){
    Client alice = Client(addrs[0], amounts[0], false);
    Client bob   = Client(addrs[1], amounts[1], false);
    _clients = [Client(alice), bob];

    _commitment = Commitment(0, 0, address(0), 0, 0, new uint256[](2));
    _state = State.Funding;
    emit VirtualBankFunding(alice.addr, alice.amount, bob.addr, bob.amount);
}
```

**源码 2:** 构造函数.

### A.1.3 资金存款

在虚拟银行的 Funding 状态，Alice 和 Bob 调用 deposit() 函数向银行转入预订额度的资金。双方的资金都转入之后，虚拟银行进入 Running 状态。双向支付通道搭建完成，Alice 和 Bob 现在可以使用 RSMC 承诺方案或者 HTLC 承诺方案进行多笔微支付。

```
/**
 * @notice Alice or Bob deposit fund to virtual bank.
 */
function deposit()  external payable isFunding() {

    if(msg.sender == _clients[0].addr
    && msg.value == _clients[0].amount
    && !_clients[0].deposited) {
        _clients[0].deposited = true;
```

```
10          emit Deposit("Alice", msg.sender, msg.value);
11      } else if (msg.sender == _clients[1].addr
12              && msg.value == _clients[1].amount
13              && !_clients[1].deposited) {
14          _clients[1].deposited = true;
15          emit Deposit("Bob", msg.sender, msg.value);
16      } else {
17          throw;
18      }
19      // If both Alice and Bob have deposited fund, virtual bank begin running.
20      if (_clients[0].deposited && _clients[1].deposited) {
21          _state = State.Running;
22          emit VirtualBankRunning(_clients[0].addr, _clients[0].amount,
23                                  _clients[1].addr, _clients[1].amount);
24      }
25  }
```

**源码 3:** 存款

## A.2 RSMC 承诺方案

RSMC 承诺方案是最简单、也是最基础的承诺方案。它是 Alice 和 Bob 对于虚拟银行中的资产如何分配达成的相互承诺。虚拟银行的资产清算可以通过 RSMC 承诺方案实现。由于承诺的协商和更新只需要 Alice 和 Bob 双方参与，不需要众多的矿工参与共识，从而实现价值的传递可以快速的完成。

### A.2.1 RSMC 承诺方案数据结构

根据 §3.1.2 节所述：RSMC 承诺方案具有不可伪造性、不可篡改性、可以撤销性。RSMC 协议模拟了银行资产清算的过程。每一次支付会重新产生一份新的承诺方案，经过一段时间的积累，Alice 和 Bob 会有多份资产清算方案，但是只有最后的方案才是有效的结果。每一个 RSMC 承诺方案都有一个唯一编号。每生成一个新的承诺方案，编号加一。任何时候，编号最大的承诺是有效的，其它的历史承诺方案都已经被撤销。

如下图 20 所示，从虚拟银行建立之后，Alice 和 Bob 一共达成 N 次共同承诺，当前有效的承诺编号为 N，之前编号更低的承诺都已经被撤销。

另外注意到，每一次承诺有左右两份，互为对偶承诺，二者的资产负债表和诚信保证金的冻结时间是一样的。但是攻守双方的位置互换。左面的承诺以 Alice 为进攻方，撤销锁是 Alice 生成的，初审的签名由 Bob 签署；反之，右面的承诺以 Bob 为进攻方，撤销锁是 Bob 生成的，初审签名由

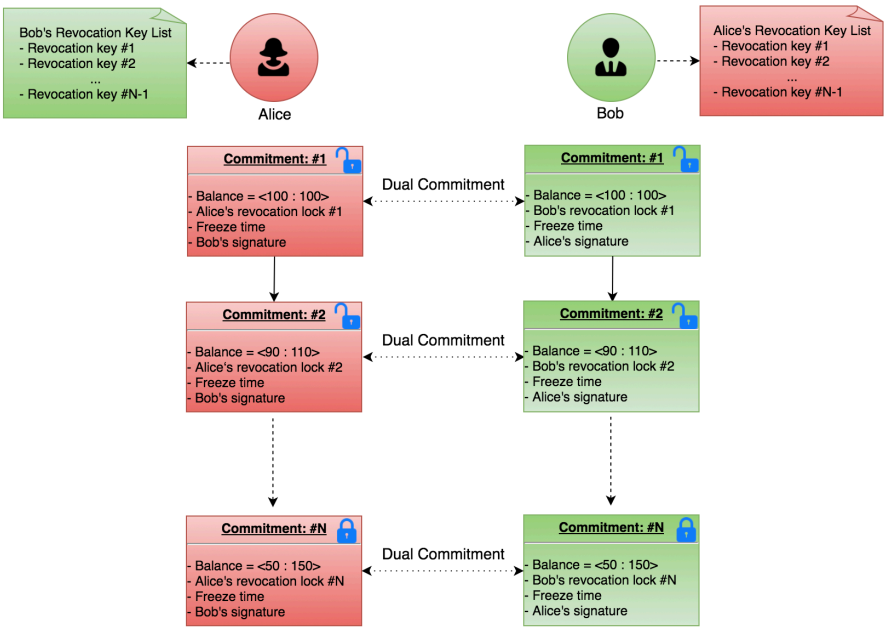


**图 20:** RSMC 承诺方案

Alice 签署。

这些承诺方案还只有防御方的签名，没有进攻方的签名，处于未兑现状态。如果某一方想兑现承诺，取出虚拟银行中的资产，可以签署自己手里的那一份承诺方案，然后广播到链上。虚拟银行智能合约收到后会按照承诺中约定的方式结算双方的资产。

下面按照支付通道的建立，使用，关闭三个环节介绍 RSMC 承诺方案的使用过程。

### A.2.2 创建支付通道

依然假设 Alice 和 Bob 要创建一个支付通道，要经过 4 个步骤：

1. 预备阶段：双方交换如下信息
   - 交换彼此的账户地址
   - 确定各自的出资额度: 比如 Alice 计划出资 100 美元, Bob 也出资 100 美元。
   - 协商诚信保证金的锁定时间 (FreezeTime)
   - 提前交换一批撤销锁。比如先交换编号 1-100 的撤销锁。只是交换撤销锁地址，对应的私钥先不要交换。
2. 创建初始的 RSMC 共同承诺：根据已经约定的出资额度 [100, 100], 双方彼此签署第一份 RSMC 承诺方案，编号为 #1. 以上图为例：
   - Alice 以防御方的身份，为右侧 1 号承诺方案签名，然后发给 Bob。
   - Bob 以防御方的身份，为左侧 1 号承诺方案签名，然后发给 Alice。
3. 建立虚拟银行智能合约：Alice 或者 Bob 部署虚拟银行智能合约，其中的配置参数是双方的地址，和出资额度。

4. 分别向虚拟银行注资：虚拟银行智能合约的地址确定后，双方分别向虚拟银行注资。

至此虚拟银行的状态为 Running，Alice 和 Bob 之间建立了双向支付通道。

### A.2.3 更新承诺方案

假如当前的承诺编号是 1，对应的资产分配方案是 [100, 100]，如果 Alice 要向 Bob 支付 10 美元，双方的资金需要按照 [90, 100] 达成一个新的承诺方案。这个过程分为两步：

1. 生成新的 RSMC 共同承诺：根据已经约定的出资额度 [90, 110], 双方彼此签署第一份 RSMC 承诺方案，编号为 #2. 以上图为例：
   - Alice 以防御方的身份，使用 Bob 的 2 号撤销锁，为右侧 2 号承诺方案签名，然后发给 Bob。
   - Bob 以防御方的身份，使用 Alice 的 2 号撤销锁，为左侧 2 号承诺方案签名，然后发给 Alice。
2. 撤销编号为 1 的承诺方案：双方互相公布对应的撤销锁私钥
   - Alice 把左侧 1 号承诺方案撤销锁的私钥发给 Bob，表示已经放弃该承诺。
   - Bob 把左侧 1 号承诺方案撤销锁的私钥发给 Alice，表示已经放弃该承诺。

如此往复，每一次支付承诺编号加一，而且换一对新的撤销锁。一直到达成第 N 个共同承诺，如下图所示。前面的 N-1 个共同承诺已经撤销，现在双方只能提交编号为 N 的共同承诺方案。因为 Alice 存储着 Bob 的前 N-1 个撤销锁私钥；同时 Bob 存储着 Alice 的前 N-1 个撤销锁私钥。如果任何一方提交了已经被撤销的承诺方案，那么对方就可以破解撤销锁，取走诚信保证金。

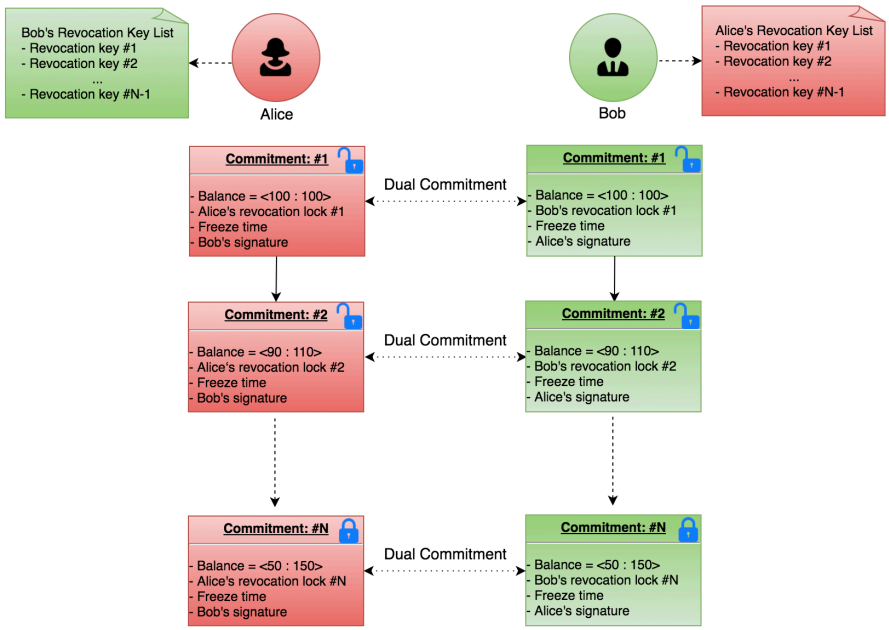


**图 21:** RSMC 承诺方案

### A.2.4 承诺方案兑现，关闭支付通道

达成共同承诺之后，任何一方可以选择在任何时候向虚拟银行提交最新的承诺方案，请求兑现双方的资产。还是接着上面的例子来讲。假设 Alice 要左侧提交编号为 N 的承诺方案，其中资产分配方案为: [50, 150]。此方案里已经有了 Bob 的签名，Alice 需要再加上自己的签名就可以广播的网络上，调用虚拟银行智能合约中的 cashRsmc() 方法，请求最终结算资产。

下面是 cashRsmc() 函数的源码，它接受 5 个参数，分别是：

- sequence: 承诺方案的编号
- balances：Alice 和 Bob 最终的资产分配方案，此例中为：[50, 150]
- revocationLock: 编号为 N，进攻方为 Alice 的撤销锁
- freezeTime: 诚信保证金的冻结时间
- defenderSignature：防御方，也就是 Bob 对此承诺方案的签名。证明 Bob 认可此承诺方案的所有配置。

rsmc() 首先检查所有输入参数是否有效，包括以下内容：

- 当前虚拟银行出于 Running 状态
- 撤销锁不是黑洞地址
- 资产总额保持一致
- 识别进攻方的身份，并且验证防御方的签名

```
/**
 * @notice Virtual bank cash a RSMC commitment which is submitted by Alice or Bob.
 * @param sequence          The sequence number of the commitment.
 * @param amounts           The amounts of new balance sheet
 * @param revocationLock    The revocation lock for attacker's findelity bond.
 * @param freezeTime        The freeze time for attacker's findelity bond.
 * @param defenderSignature The defender's signature.
 */
function cashRsmc(uint32 sequence, uint256[2] amounts, address revocationLock,
                  uint freezeTime, bytes defenderSignature)
        external isRunning() validAddress(revocationLock) {
    require((amounts[0] + amounts[1]) == (_clients[0].amount + _clients[1].amount),
    "Total amount doesn't match.");
    // identify attacker's index
    uint8 attacker = findAttacker();
    uint8 defender = 1 - attacker;
    // check defender's signature over sequence, revocation lock, new balance sheet, freeze time
    bytes32 msgHash = keccak256(abi.encodePacked(address(this), sequence,
    amounts[0], amounts[1], revocationLock, freezeTime));
```

```
    require(checkSignature(msgHash, defenderSignature, _clients[defender].addr));
    uint requestTime = now;
    emit CommitmentRSMC(sequence, NAMES[attacker], amounts[0], amounts[1],
    revocationLock, requestTime, freezeTime);
    _doCommitment(sequence, attacker, amounts, revocationLock, requestTime, freezeTime);
}
```

**源码 4:** 兑现 RSMC 共同承诺

验证无误之后，调用内部函数 _doCommitment() 执行资产的分配。然后根据**防御方优先结算**的原则，先返还 Bob 的 150 美金资产。把 Alice 的 50 美元资产暂时冻结，从当前开始计算冻结时间，虚拟银行进入 Auditing 状态。

```
/**
 * @notice Virtual bank settle defender's fund immediately, and freeze the attacker's
 *         fund as fidelity bond.
 * @param sequence          The sequence number of the commitment.
 * @param attacker          The attacker's index.
 * @param amounts           Virtual bank settle fund according to this balance sheet
 * @param revocationLock    The revocation lock for attacker's findelity bond.
 * @param requestTime       The time when virtual bank recieves the commitment, ie.
 *                          the start time of fidelity bond freezing.
 * @param freezeTime        How long attacker's findelity bond will be freezed.
 */
function _doCommitment(uint32 sequence, uint8 attacker, uint256[2] amounts,
                       address revocationLock, uint requestTime, uint freezeTime)
            internal {
    _commitment.sequence = sequence;
    _commitment.attacker = attacker;
    _commitment.revocationLock = revocationLock;
    _commitment.requestTime = requestTime;
    _commitment.freezeTime = freezeTime;
    _commitment.amounts[0] = amounts[0];
    _commitment.amounts[1] = amounts[1];

    state = State.Auditing;
    emit VirtualBankAuditing();

    // send fund to defender now
```

```
    uint8 defender = 1 - attacker;
    _clients[defender].addr.send(amounts[defender]);

    emit Withdraw(sequence, NAMES[defender], _clients[defender].addr, amounts[defender]);
    emit FreezeFidelityBond(sequence, NAMES[attacker], amounts[attacker], revocationLock,
    requestTime + freezeTime);
}
```

**源码 5:** 执行共同承诺

### A.2.5 诚信保证金

前面的一章提到，设立诚信保证金的目的是防止进攻方作弊，提交已经被撤销的承诺方案。比如对于 Alice 来讲，编号为 #2 的承诺方案更有利，她可以获得 120 美元。共同承诺的兑现相当于瓜分银行的总资产，这是一种零和博弈。假设 Alice 和 Bob 都是理性的决策者，无论谁作为进攻方主动分割资产，都不会做出于对方有利，于己方不利的决策。所以在承诺方案的处理过程中，冻结进攻方的资产，同时防御方通过虚拟银行智能合约释放的事件 CommitmentRSMC() 可以得知承诺方案的编号，而且有充分的时间审查此承诺方案释放过期。

正常情况下，Alice 是诚实的，资产编号为最新的，Bob 不会对承诺方案有异议。在冻结期满之后，Alice 可以调用 withdrawByAttacker() 函数赎回被冻结的资产。此函数的源码如下。处理逻辑比较简单，首先检查虚拟银行当前状态是 Auditing, 而且此消息是由进攻方发出的。如果检查冻结时间已满，那么将诚信保证金发给进攻方，同时虚拟银行的状态变成 Closed。

```
/**
 * @notice After freezing time, attacker withdraws his fidelity bond.
 */
function withdrawByAttacker() external isAuditing() onlyAttacker(msg.sender) {
    require(now >= _commitment.requestTime + _commitment.freezeTime);
    state = State.Closed;
    emit VirtualBankClosed();

    // send fidelity bond back to attacker
    uint attacker = _commitment.attacker;
    uint256 amount = _commitment.amounts[attacker];
    msg.sender.send(amount);
    emit Withdraw(sequence, NAMES[attacker], msg.sender, amount);
}
```

**源码 6:** 锁定时间过后，进攻方取回诚信保证金

但是如果 Alice 不是诚实的，比如说她提交了编号为 #2 的承诺方案。由于 Bob 保存有编号 #1 到 #(N-1) 的所有撤销锁私钥，Bob 可以使用其中的 #2 号私钥签名，调用 withdrawByDefender() 在冻结期未满的时候取出诚信保证金。下面是 withdrawByDefender() 的源码。此函数先确定虚拟银行当前的状态是 Auditing，而且此消息是防御方发出的。然后验证撤销锁的签名是否匹配，验证成功之后将诚信保证金发给防御方。

抢先赎回需要使用豁免私钥签名，此豁免私钥是由 Alice 创建的，不同的支付对应不同的豁免私钥。在每一次支付的时候，Alice 要向 Bob 发送上一次支付对应的豁免私钥。Bob 获得此私钥之后，可以相信 Alice 已经放弃了上次支付对应的资产分配方式。具体的操作过程请参考 RSMC 协议。

```
/**
 * @notice Defender solve the revocation lock, withdraws attacker's fidelity bond as penalty.
 * @param revocationSignature  Defender's signature to open the revocation lock.
 */
function withdrawByDefender(bytes revocationSignature)
        external isAuditing() onlyDefender(msg.sender) {
    uint attacker = _commitment.attacker;
    uint defender = 1 - attacker;

    // check signature for revocation lock
    bytes32 msgHash = keccak256(abi.encodePacked(address(this), _commitment.sequence));
    require(checkSignature(msgHash, revocationSignature, _commitment.revocationLock));
    emit RevocationLockOpened( _commitment.sequence, now, _commitment.revocationLock);

    // Close virtual bank;
    state = State.Closed;
    emit VirtualBankClosed();

    // send fidelity bond to defender
    uint256 amount = _commitment.amounts[attacker];
    msg.sender.send(amount);
    emit Withdraw(sequence, NAMES[defender], msg.sender, amount);
}
```

**源码 7:** 锁定时间内，防御方取出诚信保证金

无论是 Alice，还是 Bob 取出冻结的资产，虚拟银行都进入关闭状态，对应的支付通道随之也关闭。

## A.3 HTLC 承诺方案

在 §3.3 节已经解释过 HTLC 承诺方案的含义，是在 RSCM 的基础上又增加了 Hash 锁与时间锁。下面使用 §3.3.3 节的例子，再结合虚拟银行智能合约来解释如何使用 HTLC 承诺方案在支付路径中传递价值。这个过程可以分解为 3 个阶段：

1. **准备阶段**：此阶段要确定支付路径，要链接哪些支付通道，每一段支付通道中的余额是否充足。最终支付的双方要协商 Hash 锁，并且预留足够的时间锁长度。
2. **向前传递 HTLC 承诺方案**：此阶段从支付的发送方开始，逐步在每一个支付通道建立 HTLC 承诺方案，一直到接收方为止。这个过程也是传递 Hash 锁的过程。
3. **后向传递 Hash 锁暗语**: 从接收方开始，逐个通道公开 Hash 锁的暗语，完成 HTLC 承诺方案的支付条件，并且更换新的 RSMC 承诺方案。

下面按照这三个阶段的顺序，介绍实现的细节。

### A.3.1 准备阶段

如图 22 所示，假设 Alice 要向 Carol 支付 10 美元，他们打算通过 Bob 作为中间人传递支付。Carol 生成了随机数 R，把对应的 Hash 值 H = hash(R) 发送给 Alice。

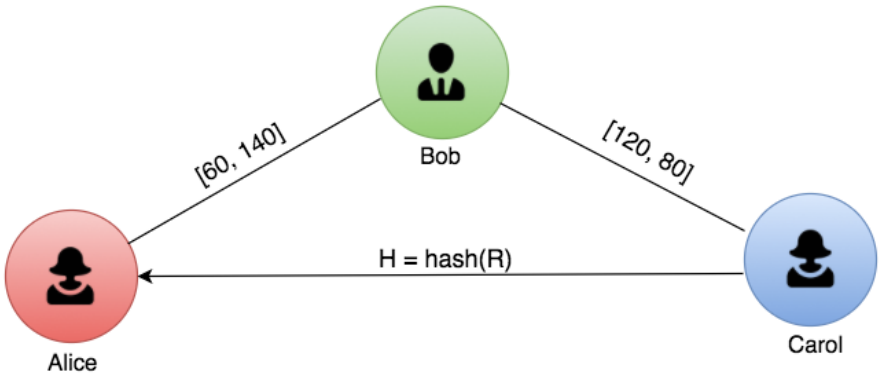


**图 22:** 支付路径

当前 Alice 和 Bob 之间的承诺分配方案的编号是 #N，资产分配方式是：[60, 140]。

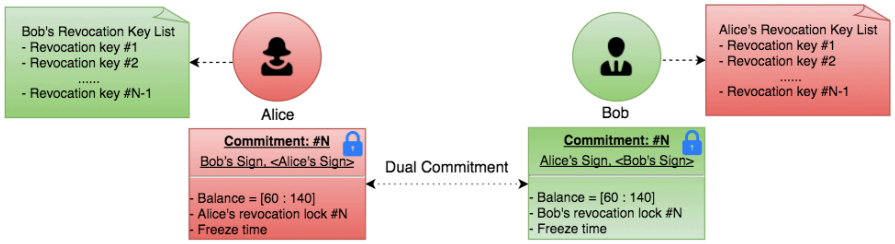


**图 23:** Alice-Bob 之间支付通道状态

同时，Bob 和 Carol 之间的承诺分配方案编号是 #M, 资产分配方式是：[120, 80]。

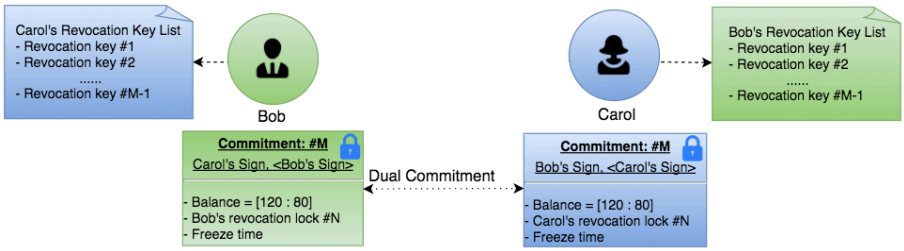


**图 24:** Bob-Carol 之间支付通道状态

### A.3.2 前向传递 HTLC 承诺方案

首先 Alice 和 Bob 先建立 HTLC 承诺方案，如下图 25 所示。新的 HTLC 方案编号为 N+1。在此承诺中，双方约定，如果 Bob 能在未来 **2 小时**之内，公开 Hash 锁对应的暗语 R，那么就按照 [50, 150] 的方式重新分配资产，等价于 Alice 向 Bob 支付 10 美元。否则还是按照 [60, 140] 的方式分配资产。同时双方都统一撤销编号为 N 的旧承诺。

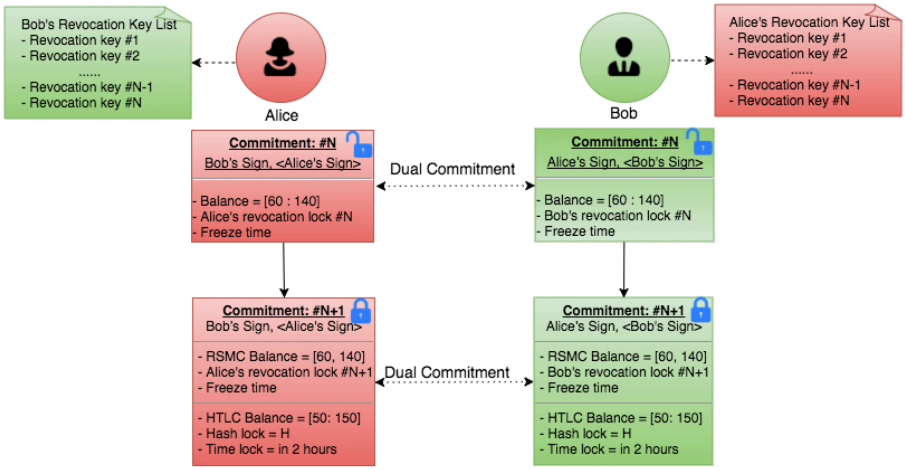


**图 25:** Alice-Bob 创建新的 HTLC 承诺方案

Bob 和 Alice 协商后，拿到 Hash 锁以及锁定的时间，然后转过身和 Carol 也建立 HTLC 承诺方案，如下图 26 所示。新的 HTLC 方案编号为 M+1。在此承诺中，双方约定，如果 Carol 能在未来 **1 小时**之内，公开 Hash 锁对应的暗语 R，那么就按照 [110, 90] 的方式重新分配资产，等价于 Bob 向 Carol 支付了 10 美元。否则还是按照 [120, 90] 的方式分配资产。同时双方都统一撤销编号为 M 的旧承诺。

需要注意的是，这里的时间锁为 1 小时，前一个时间锁为 2 小时。这是为了保证 Bob 从 Carol 那里获得 Hash 锁的暗语 R 之后，有足够长的时间在传递给 Alice。否则一旦逾期，就算拿到 R 也没有意义了。

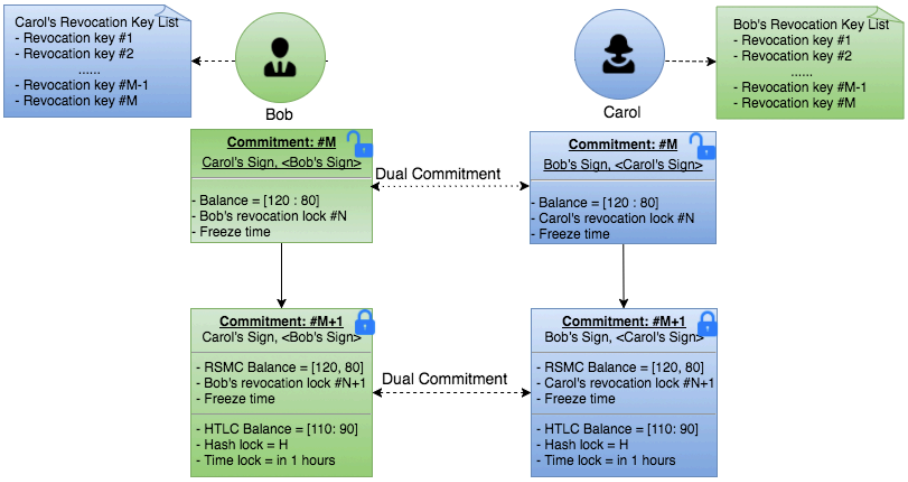


**图 26:** Bob-Carol 创建新的 HTLC 承诺方案

### A.3.3 后向传递 Hash 锁暗语

由于 Carol 知道暗语 R，在约定的 1 小时之内把 R 的值公开给 Bob。Carol 有两种方式公开 R。一是私下里把 R 发送给 Bob，第二种是把 HTLC 承诺方案 #M+1 提交给虚拟银行结算，由虚拟银行公布 R 的值。

在第一种情况下，Bob 和 Carol 都认可新的资产分配方案 [120, 80]，他们可以再创建一个长期有效的 RSMC 承诺方案 #M+2，同时撤销临时性的 HTLC 承诺方案 #M+1，如下图 27 所示。

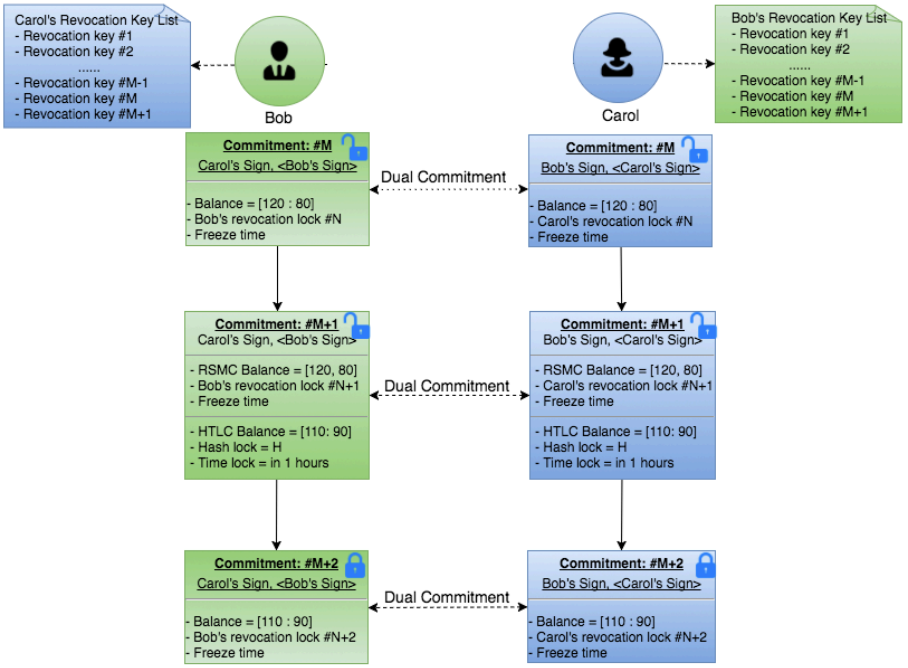


**图 27:** Carol 公开暗语 R，然后创建无期限的 RSMC 承诺方案替代临时的 HTLC 承诺方案

如果是第二种情况，Carol 在 HTLC 承诺方案 #M+1 中补上自己的签名，然后调用虚拟银行智

能合约的 cashHtlc() 方法，请求虚拟银行结算。对应的源代码如下。cashHtlc() 函数会检查时间锁是否预期，如果预期那么按照原来的 [120, 80] 结算资产。如果没有预期，而且 Hash 锁也匹配，那么按照新的 [110, 90] 结算资产。

```
/**
 * @notice Virtual bank cash a HTLC commitment which is submitted by Alice or Bob.
 * @param sequence          The sequence number of the commitment.
 * @param rsmcAmounts       Virtual bank settle fund according to this balance sheet if
 *                          HTLC time lock expire.
 * @param revocationLock    The revocation lock for attacker's findelity bond.
 * @param freezeTime        The freeze time for attacker's findelity bond.
 * @param hashLock          The hash lock in HTLC commitment.
 * @param preimage          The pre—image for the hash lock.
 * @param timeLock          The time lock in HTLC commitment.
 * @param htlcAmounts       Virtual bank settle fund according to this balance sheet if
 *                          both time lock and hash lock are satisfied.
 * @param defenderSignature The defender's signature.
 */
function cashHtlc(uint32  sequence,        uint256[2] rsmcAmounts,
                  address revocationLock,  uint       freezeTime,
                  bytes32 hashLock;        bytes      preimage;
                  uint    timeLock;        uint[2]    htlcAmounts;
                  bytes   defenderSignature)
         external isRunning() validAddress(revocationLock){

    // check rsmcAmounts
    require((rsmcAmounts[0] + rsmcAmounts[1]) == (_clients[0].balance + _clients[1].balance),
            "rsmcAmounts total amount doesn't match.");

    // check htlcAmounts
    require((htlcAmounts[0] + htlcAmounts[1]) == (_clients[0].balance + _clients[1].balance),
            "htlcAmounts total amount doesn't match.");

    // identify attacker's index
    uint8 attacker = findAttacker();
    uint8 defender = 1— attacker;

```

```
34      // check defender signature over parameters
35      bytes32 msgHash = keccak256(abi.encodePacked(address(this), sequence, rsmcAmounts[0],
36                          rsmcAmounts[1], revocationLock, freezeTime, hashLock,
37                          timeLock, htlcAmounts[0], htlcAmounts[1]));
38      require(checkSignature(msgHash, defenderSignature, _clients[defender].addr));
39
40      uint requestTime = now;
41      emit CommitmentHTLC(sequence, NAMES[attacker],
42                  rsmcAmounts[0], rsmcAmounts[1], revocationLock, requestTime, freezeTime,
43                  hashLock, preimage, timeLock, htlcAmounts[0], htlcAmounts[1]);
44
45      // check time lock
46      if (requestTime >= timeLock){
47          emit TimeLockExpire(sequence, requestTime, timeLock);
48          // if time lock expire, handle this commitment as RSMC
49          _doCommitment(sequence, attacker, rsmcAmounts, revocationLock, requestTime, freezeTime);
50      } else if {
51          // check msgHash lock
52          require (keccak256(preimage) == hashLock);
53          emit HashLockOpened(address(this), sequence, hashLock, preimage, requestTime);
54          // if both time lock and hash lock are satisfied, handle this commitment as HTLC
55          _doCommitment(sequence, attacker, htlcAmounts, revocationLock, requestTime, freezeTime);
56      }
57  }
```

**源码 8:** 兑现 HTLC 共同承诺

无论是哪一种方式，Bob 都向 Carol 支付了 10 美元，同时获得了暗语 R。而且与 Alice 之间的承诺方案至少还有 1 个小时的时间，有足够的时间公示 R，从 Alice 那里获得 10 美元的补偿。同样，Bob 也有两种方式公开 R。如果选择私下里发送给 Alice，就需要再生成新的 RSMC 承诺方案 #N+2, 并且撤销 HTLC 承诺方案 #N+1，最终的结果如下图 28 所示：

至此，Alice 通过 Bob 提供的过度资金向 Carol 支付了 10 美元。整个支付过程完毕。

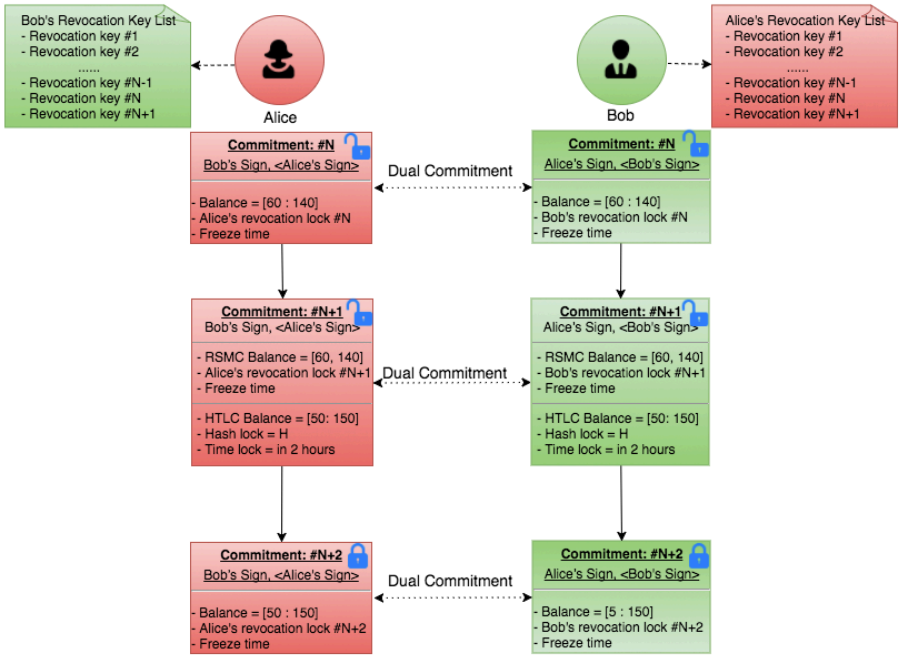


**图 28:** Bob 公开暗语 R，然后创建无期限的 RSMC 承诺方案替代临时的 HTLC 承诺方案

# 附录 B　技术的进一步拓展

自从闪电网络的白皮书公开之后，社区对其技术做了很多更加深入的研究，不断的优化去信任的实时清算协议。下面我们介绍三个相关的进展。

## B.1　Sprites 风格的 HTLC

Sprites and State Channels: Payment Networks that Go Faster than Lightning 是 2017 年发表的一篇论文，它提出了一种新的 HTLC 承诺方案。这种方案被称为 Sprites-风格的 HTLC，对应的，闪电网络协议中 HTLC 被称为闪电-风格的 HTLC。这种新的 HTLC 在 2 个方面对闪电网络协议提出了改进。

- 支持支付通道部分存取款
  在闪电网络协议中，用户一旦从虚拟银行中取款，其中的资产被全部结算，支付通道随即关闭。Sprites-风格 HTLC 支持部分取款。在不关闭虚拟银行的情况下，支付双方向可以向虚拟银行追加资金、或者部分提现。这样提高了支付通道的利用率，也节约了重新开启支付通道的成本。
- 抵押资产优化
  在闪电网络的协议中，HTLC 时间锁的大小和支付路径的长度有关系。假设支付路径的长度为 L，如下图所示，按照从接收方到支付方的顺序，时间锁的大小分别为：T，2T ... L*T。在最糟糕的情况下，资产的锁定时间随着支付路径的增加而线性增长。Sprites-风格的 HTLC 做了优化，令资产的锁定时间与路径的长度无关。

## B.2 Perun: 虚拟支付通道

在论文 Perun: Virtual Payment Hubs over Cryptocurrencies 中，又提出了一种新的支付通道链接技术，称之为“虚拟支付通道”，进一步改进了 HTLC。

这种技术扩展了虚拟银行智能合约，为支付双方提供了额外的功能。举例来说，假设 Alice 和 Carol 之间没有支付通道，但是他们分别和 Bob 有支付通道连接。在闪电网络的协议中，Bob 作为中间人要分别和 Alice 和 Carol 进行支付。但是在虚拟支付通道的协议中，Bob 并不需要确认双方的交易，甚至于 Bob 临时的离线也没有影响。虚拟通道技术可以进一步降低交易的延时和费用，同时提高了支付系统的可用性。但是要指出的是，论文中只讨论了一个中间节点的情况，如果支付路径包含多个中间节点，依然是一个开放的研究课题。

## B.3 广义状态通道 Generalized State Channels

支付通道的概念可以推广成为状态通道。在一个状态通道中，Alice 和 Bob 可以完成相对于支付更复杂的链下智能合约功能。比如说，在线游戏、资产交易等。在闪电网络中，Alice 和 Bob 在链下共同管理虚拟银行的债务分配方案。类似的，在状态通道中，双方在链下共同维护一个智能合约的状态，通过一种二元共识协议，对状态更新达成一致，不需要每次都公布到链上。任何一方都随时可以公开链下最新的状态并且同步到链上。

论文 Counterfactual: Generalized State Channels 系统的提出了广义状态通道的概念，并且为开发者提供了状态通道的开发框架，Dapp 开发者使用其 API 就能方便的集成状态通道的技术。值得注意的是，此论文提出了 **Counterfactual** 的概念，用于概括哪些行为不用上链，可以在状态通道中管理。假设一个链上的真实事件为 X，已经被矿工确认，具有不可篡改、不可伪造、不可撤销性。那么对应的链下事实为 **Counterfactual of X**，它满足 3 个条件：

1. 事件 X 还没有在链上被确认，也就是说它还没有被广播到链上。
2. 状态通道保证任何参与者都可以单方面广播 X，并且无风险的被矿工确认，成为链上事实。
3. 相关参与者的行为都假设 X 已经在链上发生了，仿佛 X 真的成为不可篡改、不可撤销的事实。

以支付通道为例，一个支付事件 X 可以是“在虚拟银行中，Alice 的账户减去 10 美元，Bob 的账户增加 10 美元”, 那么对应的 **Counterfactual of X** 可以是一个 RSMC 承诺方案：“Alice 和 Bob 共同签署一个承诺方案：Alice 的账户减去 10 美元，Bob 的账户增加 10 美元”。所以闪电网络中的承诺方案可以看成是 **Counterfactual of X** 的一个特例。

这个概念非常有价值，它不但给出了通用的状态通道的基本概念，而且也指出了链上交易的通道化的本质：构建一个可信的机制，使得智能合约的所有参与方，对于将要发生的链上事实提前达成一致，形成 **Counterfactual of X**。此事实虽然未发生，但是等同于发生，并且彼此互相信任对方，任何人都无法反悔。

# 参考文献


[1] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system (2008).* https://bitcoin.org/bitcoin.pdf.

[2] Vitalik Buterin. *Ethereum: a next generation smart contract and decentralized application platform (2013).* https://github.com/ethereum/wiki/wiki/White-Paper.

[3] David Mazières. *The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus (2016).* https://www.stellar.org/papers/stellar-consensus-protocol.pdf.

[4] *OpenZeppelin.* https://openzeppelin.org/.