# Digital Platforms and Financial Contracts*

Alexander Karaivanov

Department of Economics, Simon Fraser University

September 2025

**Abstract**

I analyze how digital platforms, combined with mechanism-design theory, can be used to implement financial transactions and multi-period state- and history-contingent contracts including payments, credit, and/or insurance. I define and model the algorithmic (code-based) tools and mechanism-design constructs used in the implementation: digital escrow, enabling payment channels; multi-signature scripts, preventing unauthorized fund use by a single party; timelocks, restricting the spending or transfer of digital assets before a set time; and token accounts, used to track history-recording variables such as promised utility. Integrating mechanism-design solutions into the platform can deal with challenges such as the users' participation, lack of trust, misrepresentation of true circumstances, and payment defaults and ensures that relevant asymmetric information or commitment obstacles are appropriately addressed so that stipulated payments are incentive-compatible and user reports are truthful. I present possible applications: an escrow-secured loan contract and a risk-sharing digital platform where users pay insurance premia or receive payouts based on reported income history.

*Correspondence address: Department of Economics, Simon Fraser University, 8888 University Drive, Burnaby, BC V5A 1S6, Canada; email: akaraiva@sfu.ca. I thank D. Andolfatto, A. Berentsen, S. Williamson, and participants at the Society for Economic Dynamics conference and the Vienna Macroeconomics workshop for excellent comments and suggestions. Financial support from the Social Sciences and Humanities Research Council of Canada (SSHRC), grants 435-2018-0111 and 435-2024-0317 is gratefully acknowledged.

# 1 Introduction

I explore the use of digital platforms to implement financial transactions and multi-period contingent contracts in economic settings. Unlike other studies that describe broad conceptual frameworks while glossing over the specifics, I provide a detailed micro-model of the building blocks and algorithmic tools of a digital platform, such as accounts, transactions, multisignature scripts, and timelocks. I then demonstrate how these digital tools and constructs can be used, together with insights and results from mechanism design theory, to implement complex multi-period state- and history-contingent financial contracts, for example a risk-sharing digital platform where users pay premia or receive payouts based on reported income history, with private information and/or limited commitment frictions.

Digital platforms and markets have grown rapidly with the recent advances and cost reductions in computing and data processing (Goldfarb and Tucker, 2019). Blockchain-based platforms, in particular, represent a fast-growing segment of financial markets. For instance, on September 20, 2025 the two leading blockchain platforms, Bitcoin and Ethereum had market capitalizations of $2.3T and $541B respectively and processed over 2M daily transactions.

These technological advances make possible digital financial platforms that leverage algorithmic tools (computer code) to collect, index, process, and store vast amounts of highly granular data, across individuals and/or over time, with privacy protection via encryption. They also enable reduced verification costs, cryptographically secure digital property rights, transfers and escrow of digital assets, and incorporating mutli-dimensional contingencies through automated "smart" contracts (Szabo 1996; Gans, 2019). While many of these tools and capabilities are not new *per se*, their implementation through digital platforms can lower costs significantly. For example, traditional escrow services typically require a bank and notary, incurring substantial fees, whereas escrow on a digital platform could achieve the same objective at a negligible cost.

At the same time, digital financial platform face important challenges, such as users' incentive to participate, lack of trust, misrepresentation of true circumstances, or default on payments. I argue that these issues can be mitigated or avoided by integrating economic theory into the platform design, specifically by applying and implementing mechanism-design solutions in which the relevant asymmetric information or commitment obstacles are appropriately incorporated and addressed and ensuring that all required payments (transfers) and user reports are incentive-compatible.

A key implementation tool is the use of digital funds as collateral or escrow, to enable one-way or two-way "payment channels", for example, in a digital insurance platform where users either make (pay premia) or receive transfers (indemnity payouts) to/from the platform. Other critical tools that I analyze include multi-signature ("multisig") scripts, which prevent a single party from

diverting digital funds and 'timelocks', a code construct that restricts the use or transfer of digital assets before a pre-specified time period has elapsed. I also show how digital token accounts can be used to track and update history-recording variables like the mechanism-design concept of promised utility.

Digital financial platforms utilizing granular data and enabling contingent incentive-compatible credit or insurance have the potential to unlock large welfare gains beyond mere transaction cost reductions. Financial inclusion is vital for improved risk sharing and mitigating poverty traps. Household-level data reveal that many people worldwide face under-insured risks related to income, health or employment.[1] These risks are especially pronounced in developing countries but also affect low-income households and small businesses globally. At the same time, access to formal financial markets and social or private insurance are limited, especially for the poorest (Banerjee and Duflo 2007). Poverty traps and economic precarity can result from self-generating series of negative income shocks and the magnitude of these risks can deter entrepreneurial activities or investments (Bowles et al. 2006, Kraay and McKenzie 2014, Ghatak 2015), ultimately depressing productivity and growth. Progress has been made in recent years, with the spread of smartphone apps and fintech,[2] however, most implementations focus on low-cost money transfers, e.g., among friends or family, and do not directly address challenges related to trust, commitment, or private information. Other examples of digital financial platforms are Tether and MakerDAO which enable collateral-based digital payments and loans (Andolfatto and Martin 2022).

Much of the early economics research on distributed ledgers, blockchain platforms and digital markets consists of overview papers (He et al., 2016; Catalini and Gans, 2016; Koeppl and Kroninck, 2017; Schar and Berentsen 2020) or studies focused on digital currency aspects (Yermack, 2014; Raskin and Yermack, 2016; Chiu and Wong, 2014; Chiu and Koeppl 2019b). In contrast, I analyze how digital platforms, together with mechanism-design theory, can be used to code and implement financial contracts including payments, credit, and/or insurance in dynamic settings with incomplete markets due to private information and/or limited commitment. I discuss and model in detail the digital-platform and mechanism-design elements and specific algorithmic tools used in the implementation.

The paper relates to and extends other recent studies that conceptualize and analyze emerging or hypothetical applications of digital platforms (mostly blockchains) and smart contracts in economic settings. These include the overview by Townsend (2020), Routledge and Zetlin-Jones (2022) on currency pegs, Holden and Malani (2021) on resolving holdup problems, Cong and He

[1] Cochrane (1991), Townsend (1994), Cole et al (2013), Samphantharak and Townsend (2018).

[2] See Jack and Suri (2014) on M-Pesa mobile phone payments in Kenya; Hau et al. (2024) and Hua and Huang (2020) on fintech and financial inclusion in China; D'Silva et al. (2019) on digital financial infrastructure as a public good in India; Nguimkeu and Okou (2021) on digital technologies in Sub-Saharan Africa, as well as Frost et al. (2019) and Croxson et al. (2023).

(2019) on enhancing entry and competition, Gans (2019) on trade, Chiu et al. (2022) and (2023) on distributed finance, Chiu and Wong (2021) on digital payments, Chiu and Koeppl (2019a) on blockchain-based settlement, Adrian et al. (2022) on currency exchange, Lee et al. (2024) on tokenized markets, Orestes and Townsend (2023) on Brazil's central bank digital currency and programmability, Aronoff and Townsend (2023) on the repo market; Townsend and Zhang (2023) on digital platforms and technologies as a central planner; and Karaivanov et al. (2023) on digital safety nets.

On the theory side, I draw on a large mechanism-design literature emphasizing the role of history dependence and promised utility as state variable in multi-period settings with private information (Spear and Srivastava, 1987; Abreu et al., 1990; Phelan and Townsend, 1991; Atkeson and Lucas, 1992; Fernandes and Phelan, 2000; Cole and Kocherlakota, 2001; Karaivanov and Townsend, 2014), or with limited commitment (Thomas and Worrall, 1988; Phelan, 1995; Kocherlakota, 1996; Ligon et al. 2000; Kehoe and Perri, 2002).

I abstract from issues related to cybersecurity or achieving consensus on decentralized platforms.[3] These technological elements, while important, are assumed to operate as intended and specified by the platform code. All following discussion about commitment and enforcement in digital transactions and contracts is predicated on this assumption and should be interpreted accordingly. I also do not address monetary (e.g., digital currency, stable coins) or regulation aspects.

# 2 Digital platform building blocks and algorithmic tools

I define and formalize key building blocks and algorithmic tools of a (blockchain-based) digital platform used for enabling transactions, payments or digital rights transfers, and collateral/escrow. To provide specifics, most of the discussion follows the Bitcoin implementation of transactions, however, the particular digital-platform implementation is not essential, both at the conceptual level and for the economic examples in Section 3.

## 2.1 Transactions

In this paper, a digital platform is defined as a database of recorded *transactions* organized in time-stamped data blocks, together with associated computer code / algorithmic rules. Cryptographic tools (e.g., Merkle-Patricia trees, hash functions) ensure the integrity of the recorded data, however, I abstract from these technological issues.

[3]Related to this is the literature on the so-called "blockchain trilemma", e.g. see Abadi and Brunnermeier (2021), refering to the trade-off between three key aspects of blockchain technology: security, scalability and decentralization.

**Transaction inputs and outputs**

Define a transaction, $\tau$ as the process of unlocking pre-existing *inputs* $x_{t-k}$ (a digital object) and locking them into *outputs* $x_t$ (another digital object). The subsripts denote time (e.g., data block number) and subscript $t-k$ where $k > 0$ reflects that the transaction inputs are pre-existing, i.e., an output of a previous transaction from time $t-k$. The leading example of a transaction, on which I elaborate below, is platform account $A$ making a digital payment to platform account $B$, in which case the inputs and outputs are digital funds (e.g., cryptocurrency or digital tokens). However, more general interpretations of inputs and outputs will be also used. A transaction can have several inputs and/or outputs.

**Cryptographic locking and unlocking scripts**

Each transaction input is locked, that is digitally secured, by a cryptographic *locking script*, for example, a cryptographic private key. Denote by $l_{t-k}$ the locking script securing given inputs $x_{t-k}$. Only the provider of a matching script/key can unlock the inputs (e.g., spend the locked pre-existing digital funds). Unlocking is the process of supplying an unlocking script, $u_t$ to the inputs of a transaction. This is called 'signing' the transaction and $u_t$ is called 'signature'.[4] If the supplied unlocking script $u_t$ matches the locking script for the inputs $x_{t-k}$, that is, if $u_t = l_{t-k}$, then the input(s) $x_{t-k}$ are unlocked and can be re-locked into output(s) $x_t$ by a new locking script $l_t$. The unlocking script can be supplied all at once or in parts (e.g., first user $A$ signs, then later on user $B$ signs, see below for further discussion and examples.)

**Definition 1: Transaction**

*A digital platform transaction $\tau$ is the mapping*

$$\tau : (\{x_{t-k}, l_{t-k}\}, u_t) \rightarrow \{x_t, l_t\}$$

*where subscripts denote time and where*

- $\{x_{t-k}, l_{t-k}\}$ *are the transaction input(s),* $x_{t-k}$ *and corresponding locking script(s),* $l_{t-k}$
- $u_t$ *is a supplied unlocking script (signature)*
- $\{x_t, l_t\}$ *are the transaction output(s),* $x_t$ *and locking script(s),* $l_t$.

For the purposes of this paper, a digital platform **account** will be identified with a private

[4] Signing a transaction asserts cryptographically that a user has the correct script or private key required to unlock/use the referenced inputs but does not reveal that script/key.

cryptographic key denoted $k^a$.[5] Denote by $\mathcal{T}$ the set of all transactions $\{\tau_t^i\}_{i,t}$ on the platform where $i$ indexes the transaction and $t$ indexes its time stamp, up to the current date. These data can be used to trace all past transfers or balances, e.g., by sender or recipient. Account balances can be derived from the record of all transactions as the sum of all funds unlockable by an account's private key (e.g., in Bitcoin); or are retrievable from the platform data as part of the virtual machine state (e.g., in Ethereum).

### 2.1.1 Locking scripts

Locking scripts secure and establish ownership (property rights) over digital funds or assets by cryptographic tools and computer code.[6] A locking script, e.g., a private cryptographic key, cannot be forged or guessed – it is either correct (matching) or not; this is a technological strength. Matching unlocking scripts to locking scripts (a true/false verification) as part of executing a transaction is used to transfer digital fund/asset ownership. This transfer is fully platform code-based, low-cost, and does not require third parties.

I model the following types of locking scripts $l$:

(a) **simple key**

$$l_{t-k} = k^a$$

where $k^a$ is a single-user cryptographic private key required to unlock/spend the transaction input(s) $x_{t-k}$.

(b) **multisignature ("multisig") script**

$$l_{t-k} = \{\textit{any } m \textit{ out of } n \textit{ keys } k^{a_1}, k^{a_2}, ..., k^{a_n} \textit{ where } m \leq n\}$$

For example, the 2-of-2 multisig script,

$$l_{t-k} = k^a \wedge k^b,$$

where $\wedge$ denotes the logical operator "and", requires that *both* of two private keys, $k^a$ and $k^b$ must be provided to unlock the inputs $x_{t-k}$. More generally, multisig locking scripts require the presentation of $m$ out of $n$ keys/signatures, therefore these scripts can be used to prevent unilateral unauthorized spending.

[5] In this paper I assume that a user is represented by a platform account and may or may not be anonymous. In general, "know your customer" or other regulatory or technological means could be used to track user identity where needed, depending on the specific platform design or application.

[6] Examples of these tools include the *scriptSig* and *scriptPubKey* functions in Bitcoin or the transaction *nonce* and *ECDSA* signature in Ethereum.

(c) **composite script**

$$l_{t-k} = s$$

In the following discussion I use two main types of composite locking scripts:

– *timelock script* – a combination of private keys $k^j$, timelocks $T^i > 0$, and the logical operators "and" $\wedge$ and "or" $\vee$; for example,

$$s_1 = (k^a, T^1) \text{ or}$$
$$s_2 = (k^a \wedge k^b, T^1) \text{ or}$$
$$s_3 = (k^a, T^1) \vee (k^b, T^2)$$

The timelock period $T^i$ is either calendar time, or number of data blocks in a blockchain-based platform, that must elapse relative to a starting point of time, for instance, the time at which the transaction was posted, for the transaction inputs to be unlockable. Example script $s_1$ above indicates that the input funds can be unlocked and used if private key $k^a$ is supplied and time $T^1$ has elapsed. Script $s_2$ also has timelock period $T^1$ but requires the multisig unlocking script, $k^a \wedge k^b$. Script $s_3$ requires either private key $k^a$ with timelock $T^1$ or private key $k^b$ with timelock $T^2$. The enforcement of timelocks is algorithmic, via the platform code, and once set they cannot be circumvented or over-ridden. Timelocks can thus be used to restrict usage rights, since timelocked funds cannot be spent by anyone before the timelock expiration, for example, supplying the correct private key $k^a$ alone cannot unlock and use funds locked by the unexpired timelock script $(k^a, T^1)$.

– *hashed timelock script* (e.g., as implemented in Bitcoin's Hashed Timelock Contract, HTLC),

$$s = H(\sigma) \vee (k^a, T)$$

where $H(\sigma)$ is the hash function value of some secret message $\sigma$.[7] Anyone with knowledge of $\sigma$ and thus $H(\sigma)$, can redeem/use the locked input funds immediately or, alternatively, a user can unlock the inputs by providing the private key $k^a$ but only after the timelock period $T$ expires. See Appendix A for an application to multi-party payments.

### 2.1.2 Creating vs. posting a transaction

In the following analysis I distinguish between

*creating a transaction* – the act of specifying the inputs, outputs, and locking and unlocking scripts of a transaction, as described in Definition 1.

[7]Hash functions $H(\sigma)$ are mathematical functions that take as input a string $\sigma$ of any length and return a value $H(\sigma)$ of fixed length. Importantly, hash functions are easy to compute but practically impossible to invert (i.e., find $\sigma$ when knowing $H(\sigma)$).

*posting a transaction* – the act of submitting the transaction for execution through the digital platform. Here I abstract from waiting time and assume for simplicity that all posted transactions are confirmed and recorded on the platform without delay.[8] Once posted and executed a transaction is irreversible.[9]

Below I assume that an interface exists, e.g., as part of the platform, through which users can pass scripts or created transactions securely to each other and where created transactions can be securely held before or without being submitted for execution on the platform. An important difference between creating and posting a transaction for execution is that the former does not involve paying platform transaction fees while the latter typically does.

### 2.1.3 Valid vs. invalid transactions

A transaction will be called *valid* if it results or would result in a successful transformation of past locked inputs into new locked outputs. A transaction will be called *invalid* if executing it on the platform results or would result in an error. Possible reasons can be that the input funds have already been unlocked and used by another transaction, or the supplied unlocking script(s) does not match the locking script(s) for all referenced inputs, or a coding error.

Importantly, a transaction $\tau$ can be invalidated on purpose by posting another transaction, $\tau'$ that unlocks and uses $\tau$'s inputs so that they would be unavailable if $\tau$ were posted for execution. Clearly, an otherwise valid transaction $\tau$ can be rendered invalid in this way only prior to being posted on the platform, i.e., by posting $\tau'$ before $\tau$. In Section 2.4 I show how this algorithmic feature can be used to implement multi-period and/or multi-party payments.

## 2.2 Algorithmic constraints

Using the notation in Definition 1, I assume that the digital platform code imposes and enforces the following constraints on the inputs, outputs, and locking/unlocking scripts for a transaction to be valid. I refer to these as 'algorithmic constraints', since they are part of the platform's computer code. The superscripts $i, j, l$ and $m$ below denote elements of the transaction inputs or outputs, $x_{t-k}$ or $x_t$.

1. **input availability** – each transaction input $x^i_{t-k}$ is an output $x^j_{t-k}$ of a previous, time

[8] In actual blockchain platforms there are submitted transactions that remain unconfirmed for some time (not recorded on the blockchain), e.g., Bitcoin's *mempool*. I abstract from this and use the term "posted" for transactions that are both submitted and confirmed.

[9] A transaction can be constructed to "reverse" the ownership of digital funds but such transaction involves a new set of inputs and outputs and would be recorded as a new transaction on the platform.

$t-k$, valid transaction

$$\forall x^i_{t-k}, \exists x^j_{t-k} \text{ such that } x^i_{t-k} = x^j_{t-k} \text{ for some } j \text{ and } t-k < t \tag{C1}$$

2. **unique input use** – no two distinct valid transactions can unlock/use the same input $x^i_{t-k}$

$$\forall x^i_{t-k}, \nexists \tau_1, \tau_2 \in \mathcal{T} \text{ with } \tau_1 \neq \tau_2 \text{ s.t. } x^i_{t-k} \text{ is an input of both } \tau_1 \text{ and } \tau_2 \tag{C2}$$

3. **proof of ownership** – for a transaction to be valid the provided unlocking script $u^i_t$ for each corresponding input must match (cryptographically satisfy) the locking script $l^j_{t-k}$ of the pre-existing transaction output, $x^j_{t-k}$ which it attempts to unlock and use,

$$u^i_t = l^j_{t-k} \text{ for the same } t, i, j, k \text{ in (C1)} \tag{C3}$$

The locking script could be *multisig* in which case two or more keys may be required to satisfy (C3).

4. **input-output funds balance** – if a transaction's inputs and outputs are digital funds (e.g., BTC, ETH), the total value of output funds $\sum_l x^l_t$ cannot exceed the total value of input funds $\sum_m x^m_{t-k}$,

$$\sum_m x^m_{t-k} \geq \sum_l x^l_t. \tag{C4}$$

Allowing strict inequality in (C4) can capture transaction fees (platform processing fees), however I abstract from modeling such fees here for simplicity.

5. **timelock enforcement** – timelocked inputs $x^i_{t-k}$ with timelock $T^i_{t-k}$ cannot be spent (transformed/locked into new outputs $x^j_t$) before the timelock period $T^i_{t-k}$ has elapsed,

$$t \geq t - k + T^i_{t-k}. \tag{C5}$$

The algorithmic constraints 1.–5. ensure that a valid transaction cannot use inputs that are not available (existing) and/or not unlockable. If an attempt is made to use funds that have already been unlocked and used by a previous posted transaction, the current transaction will be rejected by the platform as invalid.[10]

[10]For example, in Bitcoin the blockchain code and node network keep track of the complete set of unspent input funds ('UTXO set') which is updated after each recorded block of transactions. Other platforms (e.g., Ethereum) directly keep track of account balances in the platform's virtual machine state.

## 2.3 Algorithmic commitment and enforcement

The analysis in this paper focuses on algorithmic (code-based) tools to implement payments or digital ownership transfers. External/third-party commitment and enforcement technologies or institutions are assumed away. That is, any promise of a future payment or transfer must be secured and implementable by the platform code itself, in a way such that no user (including possibly under anonymity) has an incentive to deviate or renege.

Specifically, for a transaction $\tau$, as defined in Definition 1, to represent a valid commitment to make a digital transfer from user account $A$ to user account $B$ (a digital funds payment or a more general transfer of digital rights) by unlocking inputs belonging to/controlled by $A$ and re-locking them into outputs under $B$'s control, it is essential that:

**Condition C1***: the transaction input(s) $x_{t-k}$ are available and unlockable at the time of posting the transaction on the platform*.

Condition C1 requires:

(i) that the locked input(s) $x_{t-k}$ have not been already used by a previously posted valid transaction and

(ii) supplying matching unlocking script(s) $u_t$, including satisfying any timelocks if present.

An important implication of Condition C1 is that, as a prerequisite for commitment to a digital platform payment or transfer transaction, its input(s) must remain secured/locked and be available to be unlocked, via provision of the specified unlocking script, when the payment/transfer is due. In practice, this means that the transaction inputs must be secured, as in escrow.

Securing the input funds is achieved by the algorithmic tools described in Section 2.1 – multisig, timelock and combinations thereof, and by posting on the platform a special 'collateral transaction' which activates these tools and activates the digital ownership protection, thus ensuring the future availability of these inputs or funds in payments or other transactions. Multisig locking scripts, e.g., $l_{t-k} = k^a \wedge k^b$ ensure that a single party cannot spend or divert the funds without the agreement of the other party. Therefore, inputs ownership can be transferred only when the economic incentives of *both* parties are aligned, ensuring mutual trust. Timelock scripts algorithmically lock digital funds from use by *any* party and ensure that the inputs/funds are still available at the intended use/payment date. See the next section for a detailed example.

## 2.4 Payment channels

I show how the digital platform tools from Section 2.1 can be used to implement a *payment channel* (Decker and Wattenhofer, 2015), an algorithmic implementation of incentive-compatible and self-enforcing payments/transfers between two users. An extension to multiple users is briefly

presented in Appendix A.

In Section 2.3 I argued that, without external enforcement, committing to and being able to enforce future payments on a digital platform would require collateral in the form of locked funds (escrow). In addition, posting transactions on the platform may require paying transaction fees, as in Bitcoin or Ethereum for example, and there may be throughput constraints or time delays, although I abstract from those here. Consequently, an efficient implementation of digital transfers and escrow via the algorithmic tools would aim to minimize posted transactions to save on transaction costs. Below, I follow the Bitcoin implementation of payment channels, consistent with the notation developed in Section 2.1, however, this specific implementation is not essential for the main results and conclusions.

Consider a contract for payment between two agents, $A$ and $B$. To fix ideas, think of the agents starting at an initial balance state $(a_t, b_t)$ at time $t$, where $a_t$ is A's balance (digital funds) and $b_t$ is B's balance. Suppose $A$ and $B$ wish to transition to a new balance state $(a_{t+1}, b_{t+1})$. The simplest example of such balance state transition is a one-time payment $p$ from $A$ to $B$, which implies:

$$a_{t+1} = a_t - p \text{ and } b_{t+1} = b_t + p.$$

A payment channel is an incentive-compatible and self-enforcing implementation of the balance state transition described above using the algorithmic tools in Section 2.1. The key idea is to use a transaction posted on the platform to collateralize/secure the set of all possible non-negative balances $(a, b)$ between the parties, that is, the set

$$\mathcal{C} = \{\forall (a, b) \in \mathbf{R}^2_+ : a + b = c \text{ for some fixed } c > 0\}$$

where $c$ is called the payment channel capacity.

### 2.4.1 One-way payment channel

To illustrate how the algorithmic tools from Section 2.1 can be used to enforce payments, consider the simplest case of a one-way payment channel – that is, all transfers go only one way. Assume without loss of generality that $A$'s balance $a$ always decreases while $B$'s balance $b$ always increases over time. An example is a user paying for video streaming services. There are two commitment/enforcement problems to overcome: $B$ (the service provider) must be assured that she will receive the due payment(s) from $A$ (the user/buyer), and $A$ must be able to recover any escrow funds s/he has posted if $B$ disappears or reneges on the agreement. This is achieved using the algorithmic tools as follows (see Figure 1 for visual illustration).

**1. collateral/escrow set-up**. The parties create (this can be a functionality of the platform) a collateral/escrow transaction $\tau_c$ which establishes the maximum total payment that can be implemented via the payment channel,

$$\tau_c : (\{x_0, l_0\}, u_0) \rightarrow \{c, l_c\} \tag{CT}$$

The input funds $x_0$ of transaction $\tau_c$ are provided by $A$. They are locked by $A$'s private key $l_0 = k^a$ and are unlockable by supplying unlocking script $u_0 = k^a$. The value $c$ ($= \sum_j x_0^j$) is the escrow amount, establishing the payment channel's capacity. It is locked by the 2-of-2 multisig script

$$l_c = k^a \wedge k^b$$

where $k^b$ is $B$'s private key. The use of a 2-of-2 (both of two keys required) multisig locking script is essential, as it ensures that no single party can unlock and use the escrow funds without the other party's agreement (private key).

**2. refund set-up.** Before posting the collateral transaction $\tau_c$ on the platform, the parties also create a second, refund transaction $\tau_r$

$$\tau_r : (\{c, l_c\}, u_c) \rightarrow \{c, l_r\}$$

in which the escrow funds $c$ are secured by timelock $T > 0$ (e.g., 1 year in the future), that is,

$$l_r = (k^a, T)$$

This establishes the maximum duration of the payment channel. $A$ passes $\tau_r$ to $B$ to sign, i.e., to supply the $k^b$ part of the required unlocking script $u_c = k^a \wedge k^b$ (assume that the key transfer is done in a secure manner via the platform). $B$ agrees to sign, since by assumption s/he stands to benefit from the underlying contract. The refund transaction $\tau_r$, once signed by $B$ with $k^b$, protects $A$ in case $B$ stops providing the contracted service. It is is kept by $A$ and not posted on the platform (but can be posted at any time). If posted, $A$ would need to wait until the timelock period $T$ expires to access their escrow funds $c$.

**3. posting collateral/escrow.** After having $B$ sign the refund transaction $\tau_r$, user $A$ posts the collateral/escrow transaction $\tau_c$ on the platform by signing it with her private key $u_0 = k^a$. This secures the escrow funds $c$ by creating the requirement to provide the multisig script $l_c$ to unlock them and opens the payment channel. Note that *both* parties' cryptographic keys (i.e., their agreement) are necessary to unlock and use any part of the escrow funds, that is, to make a payment.

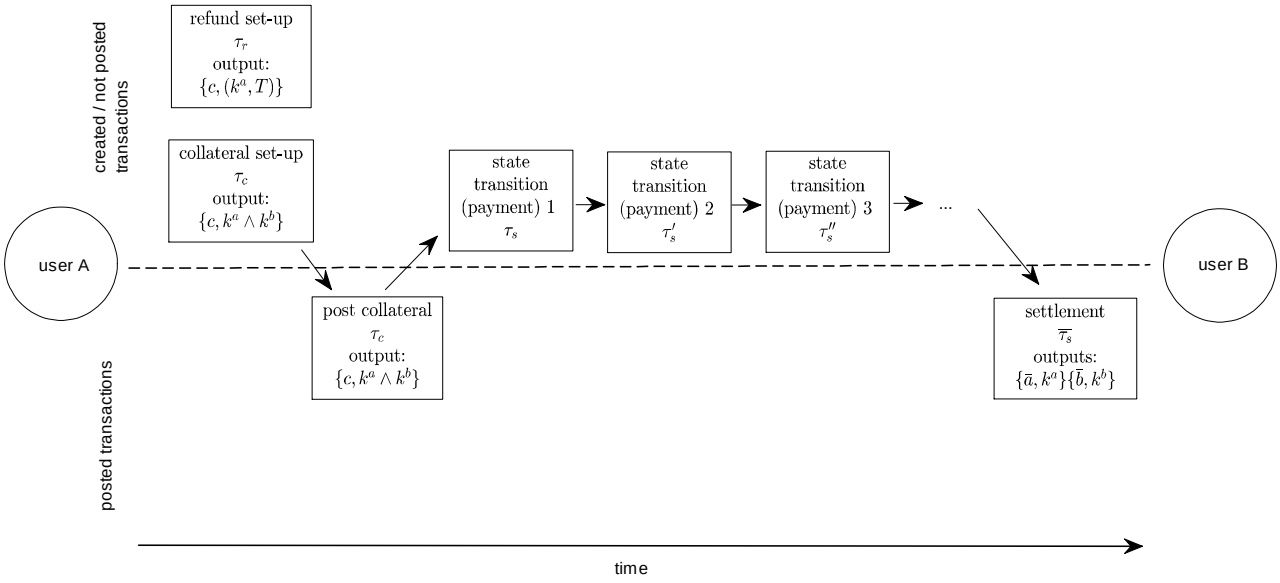


Figure 1: Payment Channel – Illustration

**4. state transition (payment).** After completing steps 1 through 3 the payment channel is established and able to perform balance state transitions of the form

$$(a, b) \rightarrow (a', b')$$

where the first argument in the brackets is $A$'s digital funds balance, the second argument is $B$'s balance, and where

$$a + b = a' + b' = c$$

In a one-way payment channel as considered here we have $b' > b$, that is, $A$ is making a digital payment (making a transfer) of size $p = b' - b > 0$ to $B$ and not vice versa.

To perform a state transition (payment), the parties create a transaction $\tau_s$ with input the escrow funds $c$ (locked by $l_c$) and two outputs, corresponding respectively to the new balances due to $A$ and $B$ respectively,

$$\tau_s : (\{c, k^a \wedge k^b\}, u_c) \rightarrow \left\{ \begin{array}{l} \{a', k^a\} \\ \{b', k^b\} \end{array} \right\} \tag{ST}$$

where $a' + b' = c$. User $A$ signs $\tau_s$ by his private key $k^a$ (supplied as part of $u_c$) and passes it to $B$. User $B$ can then post $\tau_s$ on the platform for execution by signing it with $k^b$ which would complete the required unlocking script $u_c = k^a \wedge k^b$ and would transfer $b'$ to $B$ (and $a'$ to $A$). Alternatively, $B$ could hold on to $\tau_s$, for instance, if expecting additional payments from $A$ (and hence larger balance). Further state transitions, i.e., new payments from $A$ to $B$, can be made the same way, by creating new transactions, $\tau_{s'}$, $\tau_{s''}$ etc. (this could be automated via the platform).

All these transactions have as input the locked escrow $\{c, l_c\}$ and outputs corresponding to the new balances, for example, $a''$ and $b'' = c - a''$ where $b'' > b'$ and so on.

**5. settlement.** The final state transition, or settlement, transaction $\bar{\tau}_s$ must be posted on the platform before expiry of the timelock $T$ set in step 2. Posting $\bar{\tau}_s$ closes the payment channel since the escrow funds $c$ are unlocked/used and transformed into the parties' final balances $\bar{a}$ and $\bar{b}$ with $\bar{a} + \bar{b} = c$. A new channel can be established by going back to step 1 (e.g., if the user subscribes to the video streaming platform again).

Observe that only two transactions are posted on the platform (and would incur transaction fees) – the collateral/escrow transaction $\tau_c$ and the settlement transaction $\bar{\tau}_s$, however, multiple payments from $A$ to $B$ can be completed in the meantime.

**Result 1**. *The one-way payment channel state transitions (payments) implemented by transactions $\tau_c$, $\tau_r$ and $\tau_s$ defined above are incentive-compatible and self-enforcing.*

All payments are secured and made incentive-compatible via the algorithmic tools from Section 2.1 without external enforcement, including when the users $A$ and $B$ are anonymous (their real identity is unknown):

– the *multisig* locking script $l_c = k^a \wedge k^b$ used in $\tau_c$ and $\tau_s$ ensures that neither party can unilaterally unlock and use the escrow funds $c$. This protects $B$, the seller in the video streaming example, by preventing $A$ (the buyer in the example) from posting an old balance state (i.e., old transaction $\tau_s$) that gives $A$ larger balance than that of the most current state (e.g., not making a payment for $B$'s services). Specifically, $A$ is algorithmically prevented from posting such old transaction (algorithmic enforcement) by not having $B$'s cryptographic key to unlock the escrowed inputs.

– at any time along the payments chain, $B$ has no incentive to post an old payment transaction $\tau_s$ since $B$ is due a larger balance at each following state (e.g., $b'' > b'$).

– the *timelock* period $T$ in the refund transaction $\tau_r$'s locking script $l_r = (k^a, T)$ protects $A$ (the buyer) by ensuring that s/he can eventually get their escrow funds $c$ back if $B$ (the seller) unilaterally quits and does not post a settlement transaction $\bar{\tau}_s$. The timelock script also protects $B$, by giving her sufficient time to post a settlement transaction (the latest transaction $\tau_s$ that $B$ has in possession) in case $A$ quits or disappears.

### 2.4.2 Bilateral payment channel

The one-way payment channel described in Section 2.4.1 is only incentive-compatible and self-enforcing when the same party, e.g., $A$ is always paying the other party ($B$), that is, when more and more of the escrow funds $c$ become due to $B$ over time. To see this, suppose at some point along the

payment chain the parties wanted to perform the reverse-direction state transition $(a, b) \to (a', b')$ where $b > b'$, i.e., $B$ making a payment $b - b'$ to $A$. In such case nothing would prevent $B$ from posting an old transaction $\tau_s$ already signed by $A$ (i.e., the transaction giving $B$ control of the larger previous balance, $b$) and benefit, to $A$'s loss. To rule out such incentives to deviate and to enable payments in either direction via the payment channel, additional steps and transactions utilizing the Section 2.1 algorithmic tools are required, as defined and described below.

Formally, a bilateral payment channel supporting state transitions $(a, b) \to (a', b')$ with $a + b = a' + b'$ and either $b > b'$ or $b < b'$ (e.g., $A$ pays $B$ or $B$ pays $A$) is implementable using the algorithmic tools as follows:

**b1. collateral/escrow set-up and posting.** First, the parties create and post on the platform the following escrow transaction:

$$\tilde{\tau}_c : (\{a_0, k^a; b_0, k^b\}, u_c) \to \{c, l_c\}$$

The transaction $\tilde{\tau}_c$ has two inputs, originating respectively from $A$ and $B$: $a_0$ locked with $A$'s private key $k^a$ and $b_0$ locked with $B$'s key $k^b$. The total collateral/escrow funds are $c = a_0 + b_0$. As in the one-way payment channel, the escrow transaction has a single output with value $c$ locked by the multisig key $l_c = k^a \wedge k^b$, requiring both parties' private keys.

**b2. counter-signed transactions and revocation scripts.** Suppose $A$ and $B$ would like to perform a transition from balance state $(a, b)$ to a new state $(a', b')$ with $a + b = a' + b' = c$, where initially $a = a_0$ and $b = b_0$ and where either $b' > b$ or $b' < b$ are possible. The algorithmic tools from Section 2.1 can be used to make such payments enforceable and incentive-compatible, i.e., each party is able to obtain their contracted payment without assuming trust or cooperation by the other party and without third-party enforcement beyond what is provided by the platform tools and code.

This is achieved by creating and exchanging two transactions $\tau_{sa}$ and $\tau_{sb}$, held respectively by $A$ and $B$ and counter-signed by the other party's private key, i.e., $\tau_{sa}$ is (pre-)signed by $k^b$ (indicated in (SA) as part of the unlocking script) and $\tau_{sb}$ is (pre-)signed by $k^a$:

$$\tau_{sa} : (\{c, k^a \wedge k^b\}, k^b) \to \left\{ \begin{array}{c} \{a', (k^a, T) \vee r^b\} \\ \{b', k^b\} \end{array} \right\} \tag{SA}$$

and

$$\tau_{sb} : (\{c, k^a \wedge k^b\}, k^a) \to \left\{ \begin{array}{c} \{a', k^a\} \\ \{b', (k^b, T) \vee r^a\} \end{array} \right\} \tag{SB}$$

These transactions are held by $A$ and $B$ respectively and would be valid if the required multisig

unlocking script $u_c = k^a \wedge k^b$ is completed (by signing with the other key) and they are posted on the platform.

In (SA) and (SB), the scripts $r^a$ and $r^b$, held by $A$ and $B$ respectively, are specially designed revocation scripts exchanged in each state transition step as explained below. The transactions $\tau_{sa}$ and $\tau_{sb}$ serve two roles: (i) to implement the intended state transition (payment) and (ii) as a refund guarantee, implemented via the timelock $T$.

**b3. state transitions (payments).** How do (SA) and (SB) implement payments? Transaction $\tau_{sa}$ held by $A$ has the escrow funds $c$ as its input and is pre-signed with $B$'s key $k^b$ as part of its required unlocking script $u_c = k^a \wedge k^b$. Hence, if $A$ signs transaction $\tau_{sa}$ with her cryptographic private key $k^a$ and posts it on the platform for execution, the escrow funds $c$ would be unlocked and transformed into the two outputs of $\tau_{sa}$. The output $\{b', k^b\}$ immediately transfers $B$'s current balance $b'$ into $B$'s control, unlockable by her key $k^b$. Note, however, that the other output of $\tau_{sa}$ which contains $A$'s due balance $a' = c - b'$ remains locked by the composite script,

$$s^a \equiv (k^a, T) \vee r^b.$$

This output, $\{a', s^a\}$ can be unlocked by $A$'s private key $k^a$ but only after the timelock period $T$ expires. Alternatively, noting the $\vee$ "or" operator, the balance $a'$ can be unlocked immediately by using the script $r^b$ held by $B$. Transaction $\tau_{sb}$ is analogous, exchanging the $a$ and $b$ superscripts.

To transition to a new balance state, $(a'', b'')$ the parties exchange with each other the scripts $r^a$ and $r^b$ and then create and counter-sign new transactions of the form (SA) and (SB) with input the escrow funds $c$ and outputs the new contractually due balances (again, secured by timelock and new revocation scripts). The revocation scripts $r^a$ and $r^b$ protect each agent from the other party posting an old balance state. The script exchange and counter-signing is performed algorithmically and simultaneously via the platform to avoid hold up. Finally, settlement is done by posting the final $\tau_{sa}$ and $\tau_{sb}$ on the platform before the timelock expiry.

**Result 2**. *The bilateral payment channel state transitions (payments) implemented by transactions $\tilde{\tau}_c$, $\tau_{sa}$ and $\tau_{sb}$ defined above are incentive-compatible and self-enforcing.*

To see that, suppose $A$ attempted to renege on a due payment (currently agreed balance state) and instead posted, by signing with his key $k^a$, a transaction $\tau_{sa}^{-1}$ corresponding to an old balance state which gives $A$ larger balance, $a^{-1}$ than currently due. In such case $B$ would receive $b^{-1} = c - a^{-1}$ immediately and would have $T$ periods (the timelock on $A$'s output in $\tau_{sa}^{-1}$, see (SA)) to detect $A$'s deviation and use $\tau_{sa}^{-1}$'s revocation script $r^b$ in his possession to unlock and claim the output $a^{-1}$ too, essentially seizing $A$'s share of the escrow funds $c$ and punishing the deviation. The same argument applies for $B$ in the symmetrically opposite situation.

Because of the algorithmically-enabled penalties secured by the escrow funds $c$ neither party has an incentive to post an old transaction. However, what if a party disappears, e.g., does not exchange a revocation script or does not counter-sign the next state transition transaction as planned? In that case the other party can still post the last signed transaction in their possession after waiting for its timelock to expire and therefore is protected against such deviation too.

# 3 Applications

I now present two economic applications: a loan contract and a multiperiod risk-sharing setting with private information and/or limited commitment. I discuss the use of the digital-platform algorithmic tools from Section 2 and develop specific implementation strategies for addressing the relevant enforcement or information frictions in these financial contract settings.

## 3.1 Digital loan contract

### 3.1.1 Setting

Suppose a borrower $A$ needs a fixed-term (e.g., one year) loan of size $l$ denominated in digital funds. The borrower enters a contract with a digital platform lender $B$ which agrees to supply the loan at interest rate $r$. That is, the borrower receives $l$ at the beginning or the loan term and must repay $(1+r)l$ at the end of the loan term. I maintain the assumption of no external (courts/legal) enforcement, that is, the digital loan contract must be implementable solely via the code/algorithmic tools from Section 2, as shown below.

The key contract friction in this example is enforcement. The digital platform implementation must ensure that the loan funds are available to be disbursed when needed and that the borrower repays the loan at the agreed time.

### 3.1.2 Digital platform implementation

To implement the loan contract, each of $A$ and $B$ uses two platform accounts: an expenditure account, with balances $x^a$ and $x^b$ respectively, and a collateral/escrow account, denoted $c^a$ and $c^b$ respectively. Expenditure account balances $x^i$, $i = a, b$, are always immediately accessible (unlockable via private key $k^i$) by the respective agent $A$ or $B$, while funds in the escrow accounts are secured via multisignature and timelock scripts, as specified below.

First, the borrower $A$ provides digital escrow funds/assets $a_0$ and the lender $B$ provides escrow

$b_0$. These escrow amounts are required to satisfy

$$a_0 \geq (1+r)l \text{ and } b_0 \geq l. \tag{1}$$

The first inequality ensures that the lender can always be repaid and the second ensures that the loan is possible. The escrow funds are provided via a collateral transaction and are secured by the multisig script $m^{ab} \equiv k^a \wedge k^b$ and/or a timelock as described below.

For simplicity set $b_0 = l$. The initial account balances state is (using $b_0 = l$):

$$(x^a, c^a; x^b, c^b) = (0, a_0; 0, l)$$

where the escrow account $c^a$ is secured with $m^{ab}$ and a timelock $T$ synchronized with the loan term (maturity date).

Making the loan is implemented via a disbursement transaction in which the loan amount $l$ is unlocked and debited from $B$'s escrow account and transferred into $A$'s expenditure account via supplying the script $m^{ab}$. The loan funds are then immediately redeemable (unlockable) by $A$ via her private key $k^a$ which results in account state

$$(l, a_0; 0, 0).$$

The borrower $A$ can then withdraw and use the loan funds $l$, resulting in balance state $(0, a_0; 0, 0)$ where the escrow $a_0$ remains secured by the multisig script $m^{ab}$ and the timelock $T$ synchronized with the loan term.

The loan repayment is implemented via posting a time-delayed (using a timelock) settlement transaction $\tau_s$ backed by the escrow funds $a_0$ which debits the due repayment $(1+r)l$ from $A$'s combined account balance ($x^a + c^a$) at the loan maturity date, prioritizing the current account funds (if any). This is always possible since $a_0 \geq (1+r)l$ by (1). If $A$ intends to repay the loan as agreed, she adds $(1+r)l$ to her expenditure account before the repayment is due and thus when the timelock $T$ expires $\tau_s$ automatically unlocks and transfers the due amount $(1+r)l$ to the lender's account resulting in account state

$$(0, a_0; (1+r)l, 0).$$

At this time the collateral $a_0$ is also unlocked by the timelock $T$ expiry and becomes available to $A$, to secure a new loan if needed. Essentially, by repaying as agreed, the borrower $A$ recovers her collateral while the lender $B$ recovers their initial investment $b_0 = l$ and gains the interest amount $rl$.

If, instead, the borrower $A$ fails to repay the loan at the specified time $T$ (e.g., disappears),

the time-delayed settlement transaction $\tau_s$ backed by the escrow $a_0$ outputs the following account end-balances for the two parties,

$$(0, a_0 - (1+r)l; (1+r)l, 0)$$

essentially liquidating (part of) $A$'s posted collateral to repay the lender.

A real-world example of such digital platform loan is MakerDAO's borrowing facility implemented on the Ethereum blockchain. A user locks Ethereum cryptocurrency (ETH) as collateral and receives DAI digital token denominated loan at a minimum 1.5-to-1 collateral to loan ratio, e.g., $150 worth of ETH maps to $100 worth of DAI. If the loan is repaid, the ETH collateral is released back to the borrower; if not, the collateral is liquidated in favor of the platform (the lender).

## 3.2 Risk sharing with private information (digital insurance)

### 3.2.1 Setting

A continuum of ex-ante identical risk-averse agents with preferences over consumption $u(c)$ and discount factor $\beta \in (0,1)$ face a stochastic income process $\{y_t\}_{t=0}^{T}$ which takes discrete values $y_s$, $s = 1, ..., S$ with respective probabilities $\pi_s$ where $\pi_s > 0$ and $\sum_{s=1}^{S} \pi_s = 1$. The agents have access to a risk-neutral digital insurance platform. The income realizations are private information.[11]

Using standard arguments, the optimal multi-period contracting problem in this setting can be written recursively in terms of the agent's promised utility $w$ (present value of future utility), as the state variable. The contract-theoretic construct of promised utility can be interpreted as a user's 'experience score' or 'credit score', a history-tracking variable.

The insurer requires each user to self-report their income. The platform's problem is to design a multi-period insurance contract consisting of the following two endogenous and optimally determined (see below) components:

1. *a transfer function,* $\tau(y, w)$ determining the due insurance contribution (premium) or insurance payout, where the transfer amount received by user $i$ at time $t$ given income report $y_t^i$ and current state (experience/credit score) $w_t^i$ is

$$\tau_t^i = \tau(y_t^i, w_t^i) \tag{2}$$

[11] In this setting, Townsend (1982) shows how a multi-period insurance contract that conditions risk-sharing transfers on the history of income reports dominates a debt contract in terms of efficiency.

2. *a score update rule,* $\phi(y, w)$ for the state variable $w$

$$w^i_{t+1} = \phi(y^i_t, w^i_t) \tag{3}$$

Negative values of the transfer $\tau^i_t$ mean that the user is making a contribution to the platform (paying an insurance premium) while a positive $\tau^i_t$ value means that the user is receiving a payout (insurance indemnity) from the platform. Incentives for truth-telling are provided via the current transfer $\tau^i_t$ and the future due transfers (positive or negative) encoded in the function $\phi$, an endogenous object to be solved for.

The timing is as follows:

(i) an agent's income $y^i_t$ is realized;

(ii) the agent makes a report of their current income, $\tilde{y}^i_t$. S/he can consider misreporting at this time, i.e., report $\tilde{y}^i_t \neq y^i_t$;

(iii) given the user's income report and current score $w^i_t$, an insurance transfer $\tau^i_t$ (positive/payout or negative/contribution) is computed in (2) and enacted, to or from the platform;

(iv) the next-period promised utility / score, $\phi$ is computed and recorded using (3).

For now, assume for simplicity that the agent cannot renege on a due contribution and the only friction is the private information. An extension allowing for limited commitment/imperfect enforcement is presented afterwards. The constrained optimal multi-period risk-sharing contract for a user with current score (promised utility) $w$ and income $y_s$ solves the following dynamic programming problem:

$$\Pi(w) = \max_{\{\tau(y_s,w),\phi(y_s,w)\}} \sum_s \pi_s[-\tau(y_s, w) + \frac{1}{R}\Pi(\phi(y_s, w))] \text{ subject to:}$$

$$\sum_s \pi_s[u(y_s + \tau(y_s, w)) + \beta\phi(y_s, w)] = w \text{ [promise keeping, PK]}$$

$$u(y_s + \tau(y_s, w)) + \beta\phi(y_s, w) \geq u(y_s + \tau(\tilde{y}_s, w)) + \beta\phi(\tilde{y}_s, w) \ \forall s, \forall \tilde{y}_s \neq y_s \text{ [truth-telling, TT]}$$

Above, the function $\Pi(w)$ is the insurance platform's profit (value) function and $\frac{1}{R}$ is the platform's discount factor. The initial value of $w$ is set so that $\Pi(w_0) = 0$, i.e., the platform breaks even in expectation, that is, provides actuarially fair (although not full, because of the information friction) insurance. Constraint (TT) ensures the incentive-compatibility of truth-telling, i.e., $\tilde{y}_s = y_s$ at optimum. For any given sequence of income realizations/reports for each user, the functions $\tau$ and $\phi$ are used to compute the optimal insurance transfers (contributions or payouts) and update the user's credit score as per (2) and (3). At any time and after any history constraint (PK) ensures that the future transfers and credit score (the next period's promised utility, $\phi(y_s, w)$) are consistent

with the current user score (promised utility), $w$. The dynamic programming formulation essentially allows the optimal transfers $\{\tau_t^i\}_{t=1}^T$ to be a function of the complete history of user reports about their past income realizations $y_{j_1}, y_{j_2}, ...$ up to any period $T$ with $y_{j_t} = y_s$

$$\tau(y_s, w) = f(y_{j_1}, ..., y_{j_{T-1}}, y_s).$$

Without the private information problem, the first-best outcome in this setting would be full insurance, i.e., the user receives constant consumption $c_s = \bar{c}$ in all income states $s = 1, ...S$, by receiving or making transfer $\tau_s^{fb} = \bar{c} - y_s$. The first-best outcome is, however, not incentive-compatible when the user's income $y_s$ is private information, since the user would have an incentive to report the income level that results in the largest payout (e.g., the lowest $y_s$). The truth-telling constrains (TT) are therefore imposed on the risk-sharing transfers and future promised utility $\phi$ to ensure, using the Revelation principle, that the user would optimally report her true income, $y_s$ and receive the on-path transfer (contribution or payout) $\tau(y_s, w)$ and new promised utility / score $\phi(y_s, w)$, as opposed to reporting some other income $\tilde{y}_s \neq y_s$ and receiving transfer $\tau(\tilde{y}_s, w)$ and score $\phi(\tilde{y}_s, w)$.

### 3.2.2 Digital platform implementation

The constrained optimal risk-sharing contract can be implemented on a digital platform using the following algorithmic tools. First, an off-platform module is used to solve the dynamic programming problem and determine the optimal risk-sharing transfers (contributions or payouts) for any history of income reports and based on a known (calibrated or estimated) income process $\{y_s, \pi_s\}$ and other model parameters, e.g., for the user's preferences. The same offline module also computes the user credit/experience score (promised utility) update rule $\phi$. The digital platform collects and records all income reports and score data, and implements the computed optimal mechanism-design risk-sharing transfers (insurance contributions/premia or payouts/indemnities) for all users depending on their income history.

As in Section 3.1, an expenditure and escrow account denominated in digital funds are created on the platform for each user. The platform also has an aggregate escrow account. Each user is also assigned a virtual token account with no monetary value in which the credit/experience score value $w$ (the promised utility in the mechanism-design solution) is recorded and stored as it evolves over time. The initial value of the credit score, $w_0$ is set so that $\Pi(w_0) = 0$ for all users, i.e., so the risk-sharing platform breaks even in expectation, over the long-term, with each user and thus overall as well.

In each period $t \geq 0$, user $i$ with current score $w_t^i$ makes an income report, $y_t^i$. The income

report is fed into the offline solution of the dynamic mechanism-design program and the optimal transfer $\tau(y_t^i, w_t^i)$ and new credit score value $\phi(y_t^i, w_t^i)$ are calculated using (2) and (3).

The resulting sequence of transfers between each user and the platform can be implemented via a payment channel, as a series of state transitions (see Section 2.4). Think of the user and the platform starting at balance state $(0, 0)$ (each has zero balance due) and initial user promised utility / score $w_0$, recorded in their digital token account. Then, given a history of the user's income $(y_{j_1}, y_{j_2}, ...)$ where $j_1, j_2, .. \in \{1, ..., S\}$ resulting in transfers $(\tau_{j_1}, \tau_{j_2}, ...)$, the following chain of state transitions is implemented:

$$(0,0) \to (\tau_{j_1}, -\tau_{j_1}) \to (\tau_{j_1} + \tau_{j_2}, -\tau_{j_1} - \tau_{j_2}) ... \to (\sum_{t=1}^{T} \tau_{j_t}, -\sum_{t=1}^{T} \tau_{j_t})$$

where $\tau_{j_t}$ denotes the optimal insurance transfer, (2) to the user (payout if positive and contribution if negative) at time $t$ if the reported income state is $j_t$. As discussed in Section 2.4, for these state transitions to be implemented via a digital payment channel, an escrow/collateral amount $c$ needs to be set up spanning the maximum possible balance due to either side. This escrow amount can be secured by the algorithmic tools from Section 2.1. It is sufficient to set

$$c = \max\{\frac{|\tau_{\min}|}{1-\beta}, \frac{|\tau_{\max}|}{1-\beta}\}$$

where $\tau_{\min}$ is the largest possible transfer (contribution) from the agent to the platform in the mechanism-design solution and $\tau_{\max}$ is the largest possible transfer from the platform to the agent in the dynamic program solution. If a user is due an insurance payout (i.e., $\tau > 0$), the amount is credited to the user's expenditure account and the user can either consume it or, if s/he wishes, move part of it to their escrow account from where funds are to be drawn when a contribution is due.

### 3.2.3 Limited commitment and enforcement

In the dynamic risk-sharing program above I assumed for simplicity that there is no commitment problem on the user side. The payment channel implementation discussed above would ensure enforceability in an automated way, via code and the digital platform tools, assuming ability to post the required escrow amount $c$ which, however, may be large.

Alternatively, the platform's mechanism design can be used to ensure by itself that a user has no incentive to renege on due transfers. Suppose that, after observing their income, the user can refuse to make a due contribution, $\tau(y_t^i, w_t^i) < 0$. This is known as a limited commitment problem (e.g., Thomas and Worrall, 1988) and the standard mechanism-design solution is to ensure that the

user has an incentive to remain in the contract (not renege) for any history of income realizations by threatening to exclude or 'blacklist' the user from the platform (e.g., in a permissioned platform) and/or close the user account.

To provide this participation incentive, it is critical to know the user's outside option if their platform account were closed. Depending on the setting, this outside option could be financial autarky (i.e., the user consumes their own income each period $c_t^i = y_t^i$), or 'saving only' (the user can only save but not borrow, as in a buffer-stock technology), or the user going to another platform. The latter possibility may include re-joining from scratch with a new account, if the platform cannot track or verify user identity (e.g., permissionless platform). A standard theoretical result is that a higher outside option value reduces the possible gains from risk-sharing in a limited commitment setting and hence a platform design that incorporates (e.g., as part of its code or rules) a credible threat of exclusion upon reneging would improve efficiency in such settings.

Denoting the value of the user's outside option by $V^{i,out}$, to ensure no reneging for all $i$ and $t$ the constrained-optimal risk-sharing contract (the endogenous functions $\tau$ and $\phi$) must satisfy:

$$u(y_t^i + \tau(y_t^i, w_t^i)) + \beta\phi(y_t^i, w_t^i) \geq u(y_t^i) + \beta V^{i,out} \tag{LC}$$

Constraint (LC) ensures that the insurance contract is self-enforcing, that is, each user has no economic incentive to renege on a due contribution at any moment of time and after any history of income realizations.

Locking in escrow (prepaid) funds $f$ as collateral can be used to relax constraint (LC). Suppose the user is required to prepay $f$ before their income is realized or, alternatively (if technologically possible) that part of the user's income is withheld in escrow and can be repossessed by the platform if the user reneges on a due transfer $\tau < 0$. This would satisfy the self-enforcement condition (LC) as long as $f$ satisfies, for all possible $y_t^i$,

$$u(y_t^i + \tilde{\tau}(y_t^i, w_t^i) - f) + \beta\phi(y_t^i, w_t^i) \geq u(y_t^i - f) + \beta V^{i,out} \tag{4}$$

Note that the due insurance premium (contribution) on the l.h.s. can be made the same as in the constrained-optimal contract by setting $\tilde{\tau}(y_t^i, w_t^i) - f = \tau_i$, however, the incentive to renege in high-income states on the r.h.s. (when $\tau(y_t^i, w_t^i) < 0$) is reduced because of the lower outside option value enabled by the escrow $f$. An appropriately chosen prepayment $f$ can thus allow improved insurance absent other obstacles or user liquidity constraints.

By having the technological ability to secure funds in escrow, store detailed granular user data, and compute complex contingent transfers, digital financial platforms can implement the optimal mechanism design history-contingent contract. The algorithmic tools used are a digital

token account to record a summary of the history of user reports (experience score / promised utility) and the cryptographic locking scripts (multisig and timelock) used to secure the collateral.

### 3.2.4 Example

Consider a numerical example illustrating the solution of the dynamic programs in the previous section. Suppose user income can take two levels: $y_L = 1$ and $y_H = 2$, with probabilities $\pi = 1/2$ and $1 - \pi = 1/2$ respectively. All users have identical risk-averse preferences represented by the utility function $u(c) = c - 0.05c^2$ and discount factor $\beta = 0.95$. The platform's discount factor is $\frac{1}{R} = 0.95$.

I compute the optimal risk-sharing transfers $\tau$ (payout if positive, contribution if negative) from the respective infinite-horizon dynamic programs, initiated at zero ex-ante profits, i.e., setting the initial promised utility/score $w_0$ to satisfy $\Pi(w_0) = 0$. Figure 1 displays the optimal transfers, $\tau$ and consumption (income plus transfer) for all 6 possible histories or length one period (2 histories) or two periods (4 histories), starting at the initial condition.[12] For example, history $L$ on the Figure means low income ($y_L$) in period 1 while history $LH$ means low income $(y_L)$ in period 1 and high income $(y_H)$ in period 2, etc. The dynamic program is computed for three different information/enforcement settings – hidden income (imposing constraint TT), limited commitment (imposing constraint LC), and both jointly (imposing both TT and LC). The mechanism-design solutions are compared to autarky (the user consumes their current income, $c_t^i = y_t^i$; denoted by a dotted red line on Figure 1) and 'full insurance' (equal consumption across all states of the world; denoted by dashed lines on Figure 1), for each one- or two-period long history.

Figure 1 demonstrates how access to insurance, despite the information and commitment frictions, significantly improves consumption smoothing over income states and over time relative to autarky. Importantly, the Figure shows that the optimal insurance transfers (contributions or payouts) are history-dependent, i.e., the exact sequence of income shocks, both in size and order, matters.

[12] The transfers for histories of any length can be computed in the same way using the dynamic program solutions for $\tau$ and $\phi$.

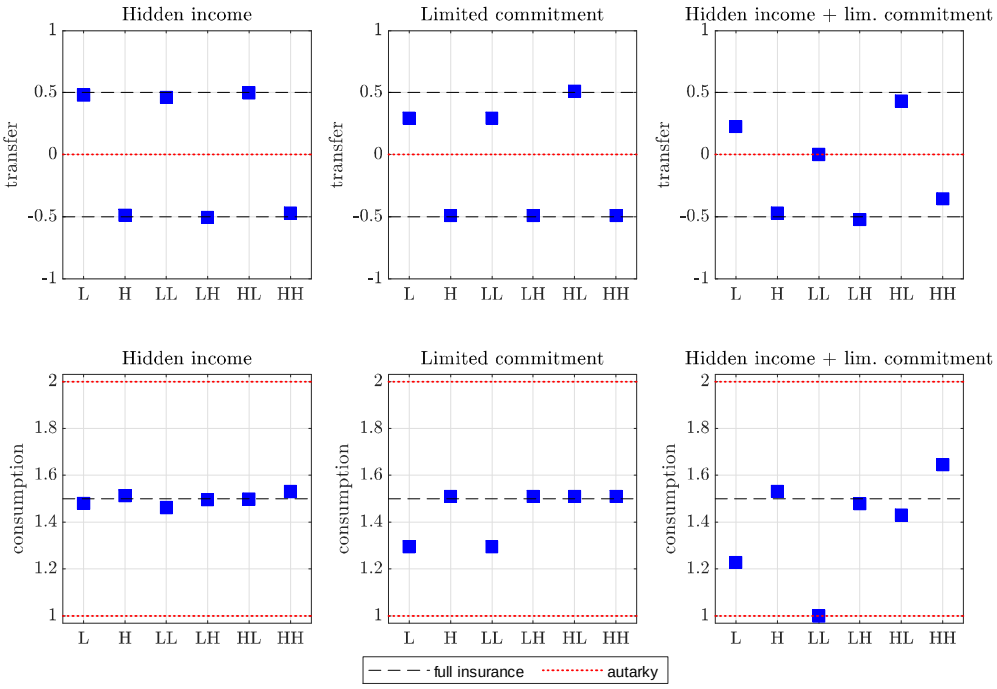


Figure 2: Optimal history-dependent transfers and consumption in multi-period contracts with financial frictions

Implementation of these mechanism-design solutions via a digital platform and the algorithmic tools discussed in the previous sections is done by recording complete data, possibly homomorphically encrypted, about the users' incomes (or their income reports in the 'hidden income' setting) which are then fed to smart contract code and mapped (via the mechanism-design dynamic program solution) to the appropriate contingent transfer to or from the platform, depending on the information or commitment/enforcement frictions present. A digital token account is used to calculate and record promised utility (experience/credit score) after each income report and the insurance transfers are function of the current income (report), or monotonic function thereof, and the promised utility token balance. Prepayment and escrow can be used to relax the limited commitment constraint as shown above.

# 4 Conclusions

Whether blockchains and other digital financial platforms are a transformative technology or merely serve as conduits for financial speculation, fraud, illicit transactions or money laundering has been an ongoing debate for years. In this paper, I contribute to this debate by describing key algorithmic building blocks, tools, and capabilities of digital platforms: construction and execution of digi-

tal financial transactions, cryptographic signatures, simple and multi-signature (multisig) locking scripts and timelocks to enforce time-based restrictions. I demonstrate that digital platforms and digital (smart) contracts offer significant potential for implementing sophisticated financial contracts. These contracts can accommodate multi-period and history-dependent contingencies and conditionalities based on current and recorded historical data in economic settings characterized by private information and limited commitment.

A central challenge, especially in decentralized blockchain-based digital platforms, is the absence of external third-party enforcement upon which traditional financial markets and intermediaries rely. To address this issue, I highlight the important role of collateral (funds held in escrow), paired with locking scripts such as multisig scripts and timelocks. These algorithmic tools secure future promised payments by ensuring that funds remain inaccessible until predefined conditions are met, thereby enabling trust and enforceability via computer code and without third-party intermediaries. I further show how escrow funds can be utilized to back multiple one-way or two-way payments (digital funds or assets transfers) through pre-established payment channels, without posting transactions on the platform, thus saving on transaction fees and waiting time. For example, a single escrow deposit can back a series of payments, such as recurring insurance premia or payouts or loan repayments, streamlining and minimizing the need for repeated on-platform settlements.

One of my main goals in this paper has been to describe some of the technological innovations in digital platforms and contracts by breaking them down to a few fundamental building blocks and algorithmic tools. Rather than using "blockchain" or "smart contract" as vague buzzwords, I provide a detailed analysis of their specific functionalities and explore their applications in realistic economic contexts. A second key goal is to advocate for the integration of economic theory, specifically mechanism-design theory, into the design and development of digital platforms, which offers a structured approach to align users' incentives, reports and actions with desired outcomes, even under asymmetric information or imperfect enforcement. Such integrated framework can guide platform developers and programmers in selecting the algorithmic tools to include in future implementations or upgrades, for instance, by creating standardized ready-to-use financial contracts such as secured loans with built-in repayment schedules or contingent insurance products that automatically compute and adjust payouts and experience/credit rating based on verified events or user reports, enhancing usability and efficiency. The implementation requires that the economic structure (income process, preferences, etc.) is known or can be estimated or learned. Structural estimation techniques (e.g., Karaivanov and Townsend, 2014) or (machine) learning algorithms can be used to calibrate the platform code using pre-existing or pilot study data.

The major technological strengths of digital platforms lie in their ability to gather and store vast

amounts of granular data, automatically perform complex contingency-based calculations using that data, and enable low-cost verification and transfer of digital funds/assets ownership. When combined with enforcement and trust mechanisms, either through platform-based escrow or, where appropriate, through trusted third parties, these capabilities can unlock substantial benefits. For example, a digital insurance platform could use real-time income data to tailor premia and payouts secured by escrow and timelocks without requiring costly manual oversight. Bringing rigorous economic theory into the design of digital platforms has the potential to reshape financial markets by improving financial inclusion, enabling more efficient risk sharing, and supporting complex automated contracts that were previously impractical.

**Disclosure of interest**

The author has no competing interests to declare.

# References


[1] Abadi, J. and M. Brunnermeier (2021), "Blockchain Economics", working paper, Princeton University.

[2] Abreu, D., D. Pearce and E. Stacchetti (1990), "Toward a Theory of Discounted Repeated Games with Imperfect Monitoring", *Econometrica* 58: 1041-63

[3] Adrian, T., F. Grinberg, T. Mancini-Griffoli, R. Townsend, and N. Zhang (2022), "A Multi-Currency Exchange and Contracting Platform", *IMF Working Paper*, November.

[4] Andolfatto, D. and F. Martin (2022), "The Blockchain Revolution: Decoding Digital Currencies", *Federal Reserve Bank of St. Louis Review* 104:149-65

[5] Aronoff, D. and R. Townsend (2023), "A Smart Contract to Increase Intermediation Capacity in the Repo Market", working paper, MIT

[6] Atkeson, A. and R. Lucas, (1992), "On Efficient Distribution with Private Information", *Review of Economic Studies* 59:427-53.

[7] Banerjee, A. and E. Duflo (2007), "The Economic Lives of the Poor", *Journal of Economic Perspectives* 21(1):141-168

[8] Bowles, S., S. Durlauf and K. Hoff, eds. (2006), *Poverty Traps*, Princeton University Press

[9] Catalini, C. and J. Gans (2016), "Some Simple Economics of the Blockchain", *NBER Working Paper* 22952

[10] Chiu, J. and T. Koeppl (2019a), "Blockchain-based Settlement for Asset Trading", *Review of Financial Studies* 32(5):1716-1753

[11] Chiu, J. and T. Koeppl (2019b), "The Economics of Cryptocurrencies: Bitcoin and Beyond", *Staff Working Paper* 2019-40, Bank of Canada

[12] Chiu, J. and T-N. Wong (2014), "E-Money: Efficiency, Stability and Optimal Policy", *Staff Working Paper* 2014-16, Bank of Canada

[13] Chiu, J. and T-N. Wong (2021), "Payments on Digital Platforms: Resiliency, Interoperability and Welfare", *Staff Working Paper* 2021-19, Bank of Canada

[14] Chiu, J., C. Kahn and T. Koeppl, (2022), "Grasping De(centralized) Fi(nance) Through the Lens of Economic Theory", *Staff Working Paper* 2022-43, Bank of Canada

[15] Chiu, J., E. Ozdenoren, K. Yuan and S. Zhang (2023), "On the Fragility of DeFi Lending", *Staff Working Paper* 2023-14, Bank of Canada

[16] Cochrane, J. (1991), "A Simple Test of Consumption Insurance", *Journal of Political Economy*, 99(5):957-976

[17] Cole, H. and N. Kocherlakota (2001), "Efficient Allocations with Hidden Income and Hidden Storage", *Review of Economic Studies* 68: 523-42

[18] Cole S., X. Giné, J. Tobacman, R. Townsend, P. Topalova, J. Vickery (2013), "Barriers to Household Risk Management: Evidence from India", *American Economic Journal: Applied Economics* 5(1):104-135

[19] Cong, L. and Z. He (2019), "Blockchain Disruption and Smart Contracts", *Review of Financial Studies* 32(5):1754-97

[20] Croxson, K., J. Frost, L. Gambacorta and T. Valletti (2023), "Platform-Based Business Models and Financial Inclusion: Policy Trade-Offs and Approaches", *Journal of Competition Law & Economics* 19:75–102

[21] D'Silva, Z., F. Packer and S. Tiwari (2019), "The Design of Digital Financial Infrastructure: Lessons from India", *BIS Papers* 106

[22] Decker, C. and R. Wattenhofer (2015), "A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels", in Pelc, A., Schwarzmann, A. (eds), *Stabilization, Safety, and Security of Distributed Systems*.

[23] Fernandes, A. and C. Phelan (2000), "A Recursive Formulation for Repeated Agency with History Dependence", *Journal of Economic Theory* 91: 223-247

[24] Frost, J., L. Gambacorta, Y. Huang, H. Shin and P. Zbinden (2019), "Bigtech in Finance", *Economic Policy*, October, 761-799

[25] Gans, J. (2019), "The Fine Print in Smart Contracts", *NBER Working Paper* 25443

[26] Ghatak, M. (2015), "Theories of Poverty Traps and Anti-Poverty Policies", *World Bank Economic Review* 29:S77–S105

[27] Goldfarb, A. and C. Tucker (2019), "Digital Economics", *Journal of Economic Literature* 57(1): 3-43.

[28] Hau H., Y. Huang, C. Lin, H. Shan, Z. Sheng and L. Wei (2024), "Fintech credit and entrepreneurial growth", *Journal of Finance* 79(5): 3309-59

[29] He, D., K. Habermeier, R. Leckow, V. Haksar, Y. Almeida, M. Kashima, N. Kyriakos-Saad, H. Oura, T. Sedik, N. Stetsenko, and C. Verdugo-Yepes (2016), "Virtual Currencies and Beyond: Initial Considerations", *IMF Staff Discussion Note* 16-03.

[30] Holden, R. and A. Malani (2021), *Can Blockchain Solve the Hold-up Problem in Contracts?*, Cambridge University Press.

[31] Hua X. and Y. Huang (2020), "Understanding China's fintech sector: development, impacts and risks", *European Journal of Finance* 27: 321-33

[32] Jack, W. and T. Suri (2014), "Risk Sharing and Transactions Costs: Evidence from Kenya's Mobile Money Revolution", *American Economic Review* 104(1):183-223

[33] Karaivanov, A. and R. Townsend (2014), "Dynamic Financial Constraints: Distinguishing Mechanism Design from Exogenously Incomplete Regimes", *Econometrica* 82:887-959

[34] Karaivanov, A., B. Mojon, L. Pereira da Silva and R. Townsend (2023), "Digital Safety Nets – A Roadmap", *BIS Papers* 139.

[35] Kehoe, P. and F. Perri (2002), "International Business Cycles with Endogenous Incomplete Markets", *Econometrica* 70: 907-28

[36] Kocherlakota, N. (1996), "Implications of Efficient Risk Sharing Without Commitment", *Review of Economic Studies* 63: 595-609

[37] Koeppl, T. and J. Kronick (2017), "Blockchain Technology – What's in Store for Canada's Economy and Financial Markets?", *Commentary No.468*, C.D. Howe Institute

[38] Kraay, A. and D. McKenzie (2014), "Do Poverty Traps Exist? Assessing the Evidence", *Journal of Economic Perspectives* 28(3):127-148

[39] Lee, M., A. Martin and R. Townsend (2024), "Optimal Design of Tokenized Markets", *Federal Reserve Bank of New York Staff Report* 1121.

[40] Ligon, E., J. Thomas and T. Worrall, (2000), "Mutual Insurance, Individual Savings and Limited Commitment", *Review of Economic Dynamics* 3:216-246.

[41] Nguimkeu, P. and Okou, C. (2021), "Leveraging digital technologies to boost productivity in the informal sector in Sub-Saharan Africa", *Review of Policy Research* 38:707-731

[42] Orestes, V. and R. Townsend (2023), "Brazil's Central Bank Digital Currency: Improving Financial Infrastructure with Programmability", working paper, MIT.

[43] Phelan, C. (1995), "Repeated Moral Hazard and One-Sided Commitment", *Journal of Economic Theory* 66:488-506.

[44] Phelan, C. and R. Townsend (1991), "Computing Multi-Period, Information-Constrained Equilibria", *Review of Economic Studies* 58: 853-81.

[45] Raskin, M. and D. Yermack (2016), "Digital Currencies, Decentralized Ledgers and the Future of Central Banking", *NBER Working Paper* 22238

[46] Routledge, B. and A. Zetlin-Jones (2022), "Currency Stability Using Blockchain Technology", *Journal of Economic Dynamics and Control* 142(104155)

[47] Samphantharak, K. and R. Townsend (2018), "Risk and Return in Village Economies", *American Economic Journal: Macroeconomics* 10(1):1-40.

[48] Schar, F. and A. Berentsen (2020), *Bitcoin, Blockchain, and Cryptoassets: A Comprehensive Introduction*, MIT Press

[49] Spear, S. and S. Srivastava (1987), "On Repeated Moral Hazard with Discounting", *Review of Economic Studies* 53: 599-617

[50] Szabo, N. (1996), "Smart Contracts: Building Blocks for Digital Markets", *Extropy* 16

[51] Thomas, J. and T. Worrall (1988), "Self-Enforcing Wage Contracts", *Review of Economic Studies* 55: 541-55

[52] Townsend, R. (1982), "Optimal Multiperiod Contracts and the Gain from Enduring Relationships under Private Information", *Journal of Political Economy* 82:1166-96

[53] Townsend, R. (1994), "Risk and Insurance in Village India", *Econometrica* 62(3): 539–91.

[54] Townsend, R. (2020), *Distributed Ledgers: Design and Regulation of Financial Infrastructure and Payment Systems*, MIT Press.

[55] Townsend, R. and N. Zhang (2023), "Technologies that Replace a Central Planner", *AEA Papers and Proceedings* 113:257-262.

[56] Yermack, D. (2014), "Is Bitcoin a Real Currency? An Economic Appraisal", *NBER Working Paper* 19747

# Appendix A. Multi-party transactions

The basic idea behind bilateral payments described in Section 2.4.2 can be extended to payment transactions involving more than two parties – an example is the Lightning Network in Bitcoin. In short, if a path of bilateral collateralized payment channels can be constructed between any two platform users (nodes) $C$ and $D$, then they can make payments to each other even if they do not have an established direct payment channel between them. The maximum transfer amount is determined by the bilateral channel with the lowest collateral (capacity) on the path.

Relying only on internal enforcement by code, via the algorithmic and cryptographic tools, the main challenge for such transfers going through possibly anonymous nodes is to guarantee that each node has an incentive to pass the digital funds or asset ownership forward and to be able to prove that it has done so. Clearly, any intermediate user/node between the end-nodes (payer and payee) $C$ and $D$ should not be able to unlock and use the funds themselves, because in such case the sender $C$ would have no recourse as, by assumption, there is no external enforcement of platform transactions.

This enforcement problem can be solved algorithmically via a *hashed timelock script* using the hash function $H(\sigma)$ of some secret message $\sigma$. A hash function is a mathematical function such that it is easy to compute $H(\sigma)$ when one knows $\sigma$ but nearly impossible to invert, that is, one cannot find $\sigma$ from knowing $H(\sigma)$. When payment is agreed upon, the recipient/payee $D$ creates the message $\sigma$ and sends its hash value $H(\sigma)$ (but, importantly, not $\sigma$ itself) to the sender/payer $C$. User $C$ then creates a transaction with output funds $p + \varepsilon$ (the agreed payment amount $p$ plus extra funds $\varepsilon$ to pay intermediary nodes, see below) which are locked by the hashed timelock script

$$l_1 = H(\sigma) \vee (k^C, T^1).$$

The payer $C$ then passes this transaction to the first intermediate node $I_1$ with whom $C$ is assumed to have a pre-established payment channel with capacity at least $p$. Note that user $I_1$ cannot redeem the locked funds since $I_1$ does not know $\sigma$ or $H(\sigma)$, but can be incentivized, for example by a small fee, to pass along the locked funds to another user (e.g., $I_2$ with whom $I_1$ has an established payment channel), with the output funds locked by the script $l_2 = H(\sigma) \vee (k^{I_1}, T^2)$ with $T^2 < T^1$, and so on. The path $I_1, I_2, ...$ and locking scripts $l_1, l_2, ...$are constructed algorithmically by the platform. The role of the timelocks $T^i$ is to ensure that each intermediate user along the payment chain would be able to receive their fee in case the secret message $\sigma$ is never provided by user $D$.

When the payee $D$ is reached, e.g., from some node $I_n$, then since $D$ knows the secret message $\sigma$, she unlocks/claims the agreed payment $p$ from her bilateral payment channel balance with user

$I_n$. Claiming $p$ algorithmically triggers sending the secret $\sigma$ to $I_n$ who in turn claims $p$ plus a small fee, $\frac{\varepsilon}{n}$ from his payment channel balance with user $I_{n-1}$ and thus benefits by $\frac{\varepsilon}{n}$. This process continues until we reach back $C$, where in the last state transition user $I_1$ claims $p + \varepsilon$ from $C$ via their bilateral payment channel. The end-result of this chain of bilateral state transitions is a payment of $p$ from $C$ to $D$, as contractually intended, with the $n$ users in between receiving $\frac{\varepsilon}{n}$ each. Note that no transactions are required to be posted on the platform in the process, as long as there are pre-existing bilateral payment channels forming a path from $C$ to $D$.