# Digital Platforms and Financial Contracts*

Alexander Karaivanov

July 2026

**Abstract**

I analyze the application of digital platforms and mechanism-design theory in implementing financial transactions and multiperiod contingent contracts for payments, credit, and insurance. I model a taxonomy of algorithmic primitives and contract theoretic constructs used in the implementation: digital escrow enabling payment channels; multi-signature scripts preventing unauthorized spending; timelocks restricting transacting before a set time; and token accounts tracking history-encoding variables like promised utility. Integrating mechanism-design solutions in the platform design can address challenges with user participation, lack of trust, default, and asymmetric information and commitment obstacles. I present possible applications: an escrow-secured digital loan and a risk-sharing 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 Vienna Macroeconomics workshop and the Society for Economic Dynamics conference for many useful 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 in financial transactions and contracts, with application to payments, credit and insurance. I provide a detailed micro-founded model of the algorithmic primitives and constraints of a digital financial platform and demonstrate how they can be integrated with mechanism-design theory to implement multiperiod history-contingent financial transactions and contracts, including repeated payments, collateralized lending, and dynamic risk-sharing.

The paper's intended audience is both academic economists studying financial contracts and fintech or policy practitioners developing digital platforms. Bridging and integrating economic theory with elements of computer science, I formalize a taxonomy of platform primitives – transactions, multisignature and timelock scripts, digital escrow, and token accounts that store and update states – and map them to contract theory constructs and solutions. I demonstrate how digital platform primitives can be designed to implement escrow-backed transfers, state updates, and payment-channel settlements for mechanisms that encode the relevant incentive and participation constraints in economic settings with limited commitment and private information frictions. I develop and show specific applications: an escrow-secured digital loan and a dynamic insurance platform where users pay premia or receive payouts based on income histories.

Digital financial markets and platforms have grown rapidly driven by technological advances and cost reductions in computing and data processing (Goldfarb and Tucker 2019). Blockchain-based platforms are a particularly fast-growing segment, with total capitalization in the trillions of dollars and processing millions of transactions daily. The technologies behind these platforms leverage algorithmic tools to collect, index, store, and process granular data over time and across individuals. In addition, they enable cryptographically secure digital property rights and verification, transfer and escrow of digital assets, and can incorporate multidimensional contingencies through automated ("smart") contracts (Szabo 1996; Gans 2019). While legacy alternatives for these functions exist, digital platforms can drastically lower costs. For example, traditional escrow services require a bank and notary involving substantial fees, whereas code-based escrow can achieve the same objective at a negligible cost.

At the same time, digital financial platforms face important challenges related to users' voluntary participation, trust, misrepresentation of true circumstances, and payment default. I argue that these challenges can be mitigated or avoided by intentionally integrating known mechanism-design solutions into the platform architecture to ensure that the relevant asymmetric information or commitment obstacles are incorporated and addressed in an incentive-compatible way.

A key implementation tool I analyze is the use of digital escrow to enable "payment channels" – an algorithmic implementation of self-enforcing payments between users. Other important technological tools include multisignature ("multisig") scripts, which prevent unilateral fund diversion,

and timelocks which restrict the use of digital funds until a pre-specified period elapses. I further show how digital token accounts can be used to track and update history-encoding variables, mapping to the mechanism-design concept of promised utility.

In terms of their broader applicability, digital financial platforms for credit and insurance have the potential to unlock large welfare gains extending beyond transaction cost reductions. Financial inclusion is crucial for improved risk sharing and mitigating economic precarity and household data consistently reveal severe under-insurance against income, health or employment shocks.[1] These uninsured risks are especially pronounced in developing countries but affect low-income households and small businesses globally. Access to formal financial markets remains very limited for the poorest (Banerjee and Duflo 2007), leaving them vulnerable to poverty traps resulting from sequences of negative shocks that can deter entrepreneurship and investment and depress productivity and growth (Bowles et al. 2006, Kraay and McKenzie 2014, Ghatak 2015).

While the recent proliferation of smartphone apps and fintech holds promise,[2] most applications focus on low-cost money transfers (e.g., among friends or family) and do not directly address deeper challenges related to trust, commitment, and private information. Other digital platform examples are Tether and MakerDAO which enable collateral-based payments and loans, demonstrating the viability of code-based financial intermediation (Andolfatto and Martin 2022).

**Related literature**

Much of the early research on digital markets and platforms consists of overview papers and broad conceptualizations (He et al. 2016; Catalini and Gans 2016; Koeppl and Kroninck 2017; Schar and Berentsen 2020; Townsend 2020) or focuses on digital currency aspects (Yermack 2014; Raskin and Yermack 2016; Chiu and Wong 2014; Chiu and Koeppl 2019b). In contrast, I formalize the code-based and mechanism-design primitives and specific implementation tools used to algorithmically execute payments, credit, and insurance in dynamic incomplete markets settings.

The current paper complements a growing literature on applications of digital platforms and smart contracts in economic settings. These include Routledge and Zetlin-Jones (2022) on currency pegs, Holden and Malani (2021) on hold-up, Cong and He (2019) on entry and competition, Gans (2019) on trade, Chiu et al. (2022, 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 digital currency and programmability, Aronoff and Townsend (2023) on the repo market; Townsend

[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 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).

and Zhang (2023) on digital technologies as a central planner; and Karaivanov et al. (2023) on digital safety nets.

On the theory side, I draw on the large mechanism-design literature on constrained-optimal recursive contracts in multiperiod settings with private information (Spear and Srivastava 1987; Phelan and Townsend 1991; Atkeson and Lucas 1992; 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).

It is important to acknowledge several technological and theoretical limitations to the proposed framework. First, I assume that all underlying technology elements function flawlessly, as specified by the platform code.[3] All results regarding algorithmic enforcement and transaction execution are predicated on this assumption. Second, the reliance on digital escrow to solve the limited commitment problem has an opportunity cost, especially with respect to low-income users who stand to benefit the most from enhanced financial inclusion. Third, I focus only on the algorithmic (internal to the platform) tools for achieving incentive-compatibility or enforcing payments, however, in practice these tools can (and likely should) be complemented by existing external information and enforcement mechanisms, where possible. The insurance platform application requires that agents' income and preference parameters can be estimated or learned and a computation module to determine the optimal risk-sharing transfers and experience score. Finally, I abstract from regulatory, legal and monetary policy issues, focusing purely on the financial contracts' micro-structure.

# 2 Digital platform primitives and algorithmic tools

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 and rules. I start by formally describing the key primitives and algorithmic tools used to implement transactions, payments, and escrow on a digital financial platform.

## 2.1 Transactions

### 2.1.1 Primitives

Define a *transaction* $\tau$ as the process of unlocking pre-existing *inputs* $x_\tau$ (a digital object) and locking them into *outputs* $y_\tau$ (another digital object). The leading example of a transaction is platform user $A$ making a digital payment to user $B$, in which case the inputs and outputs are

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

digital funds (e.g., cryptocurrency or digital tokens). More general interpretations of inputs and outputs are also possible, as described later. A transaction can have several inputs and/or outputs.

**Definition 1 (Transaction)** *A transaction is the tuple* $\tau = (I_\tau, U_\tau, O_\tau, t_\tau)$ *with*

$$I_\tau = \{(x_\tau^1, l_\tau^1), ..., (x_\tau^n, l_\tau^n)\} \quad \textit{(input set)}$$
$$U_\tau = \{u_\tau^1, ..., u_\tau^n\} \quad \textit{(unlocking scripts)}$$
$$O_\tau = \{(y_\tau^1, l_\tau'^1), ..., (y_\tau^m, l_\tau'^m)\} \quad \textit{(output set)}$$
$$t_\tau \quad \textit{(timestamp)}$$

*where*

– $\{(x_\tau^j, l_\tau^j)\}_{j=1}^n$ *are the transaction input(s),* $x_\tau^j$ *with corresponding locking script(s),* $l_\tau^j$

– $\{u_\tau^j\}_{j=1}^n$ *are supplied unlocking scripts (signatures)*

– $\{(y_\tau^k, l_\tau'^k)\}_{k=1}^m$ *are the transaction output(s),* $y_\tau^k$ *with locking script(s),* $l_\tau'^k$

– $t_\tau$ *is the transaction timestamp recording its time of executing on the platform (empty if the transaction is not yet posted).*

Each transaction input is locked, meaning digitally secured, by a *locking script* specifying the cryptographic condition required to use/transfer the input, for example, requiring a specific cryptographic private key – a secret alphanumeric code. In this paper, each platform user account is identified with such private key, denoted $k^a$.[4]

Unlocking is the process of applying an unlocking script, $u_\tau^j \in U_\tau$ (also called 'signing') to the input(s) of a transaction.[5] If the unlocking script *satisfies*, denoted $u_\tau^j \mid = l_\tau^j$, the cryptographic condition(s) set by the locking script for the input $x_\tau^j$, then the input $x_\tau^j$ is unlocked and can be re-locked into output $y_\tau^k$ using a new locking script $l_\tau'^k$. An unlocking script can be applied all at once or in parts, e.g., first user $A$ signs (satisfying part of the locking script), then user $B$ signs – see below for further discussion and examples.

Denote by $\mathcal{T}$ the set of all transactions $\{(I_\tau, U_\tau, O_\tau, t_\tau)\}$ recorded on the platform up to the current date. These data can be used to trace all past digital transfers. Platform account balances can also be obtained from $\mathcal{T}$.[6] All posted transactions are public information for all users.

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

[5] This asserts cryptographically that a user has the correct script (e.g., private key) required to unlock/use the referenced inputs but does not reveal that script or key.

[6] Balances can be constructed as the sum of all funds unlockable by an account's private key (e.g., in Bitcoin) or are retrievable from the platform virtual machine state (e.g., in Ethereum).

### 2.1.2 Types of locking scripts

Locking scripts secure and establish ownership over digital assets through cryptographic functions and code.[7] A locking script such as a private key cannot be forged – it is either correct (matching) or not; this is a technological strength. Matching unlocking scripts to locking scripts – a true/false verification – is used to transfer digital ownership. The property rights transfer is thus entirely code-based, low-cost, and does not require third parties.

In the following analysis I model the following types of locking scripts used to digitally secure funds on the platform:

(a) **private key**

$$l_\tau^j = k^a$$

where $k^a$ is a single-user cryptographic private key, e.g., required to unlock transaction input $x_\tau^j \in I_\tau$ in Definition 1. More than one input can be unlockable by the same private key.

(b) **multisignature (multisig) script** – requires or allows more than one private key,

$$l_\tau^j = \textit{any } q \leq Q \textit{ out of the } Q \textit{ private keys } k^{a_1}, k^{a_2}, ..., k^{a_Q}$$

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

$$l_\tau^j = k^a \wedge k^b,$$

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 input funds $x_\tau^j$ (dual-signature). Another example is the multisig script $k^a \vee k^b$ which requires *any* of $k^a$ or $k^b$ to unlock the input funds, e.g., as in a joint account or an expense account with multiple approvers. More generally, multisig scripts require the presentation of $q$ out of $Q$ keys/signatures,[8] and therefore can be used to prevent unauthorized unilateral spending (see Section 3 for applications).

(c) **timelock script**, $l_\tau^j = \boldsymbol{s}$ **–** defined as a combination of private keys $k^i$, timelocks $T$, and the logical operators "and" $\wedge$ and "or" $\vee$, for example:

$$\begin{aligned}
\boldsymbol{s}_1 &= (k^a, T^1) \text{ or} \\
\boldsymbol{s}_2 &= (k^a \wedge k^b, T^1) \text{ or} \\
\boldsymbol{s}_3 &= (k^a, T^1) \vee (k^b, T^2)
\end{aligned}$$

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

[8] For example, a 2-of-3 multisig script would be $l = (k^a \wedge k^b) \vee (k^a \wedge k^c) \vee (k^b \wedge k^c)$.

Timelock scripts specify a time period $T$ that must elapse, relative to a defined starting point, for the inputs to be unlockable. For example, script $\boldsymbol{s}_1$ above indicates that the input funds can be unlocked if private key $k^a$ is supplied and time $T^1$ has elapsed. Script $\boldsymbol{s}_2$ also has timelock period $T^1$ and requires the multisig unlocking script $k^a \wedge k^b$. Script $\boldsymbol{s}_3$ 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 code, and once set they cannot be circumvented or overridden. 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 $(k^a, T^1)$.

### 2.1.3 Algorithmic constraints

The digital platform code imposes and enforces the following constraints, C1–C5, on the inputs, outputs, and locking/unlocking scripts to ensure that a transaction, as defined in Definition 1, is *valid*, that is, can be successfully executed on the platform. I call these 'algorithmic constraints' since they are part of the platform code.

**Definition 2 (Valid transaction).** *A transaction $\tau = (I_\tau, U_\tau, O_\tau, t_\tau)$ is **valid** if and only if all of the following constraints are satisfied:*

**C1. input availability** – *for every transaction input $(x, l) \in I_\tau$ there exists a prior transaction $\tau' = (I_{\tau'}, U_{\tau'}, O_{\tau'}, t_{\tau'}) \in \mathcal{T}$ with output $(y, l')$ and timestamp $t_{\tau'} < t_\tau$ such that*

$$(x, l) = (y, l') \in O_{\tau'} \tag{C1}$$

**C2**. **unique input use** – *no two distinct valid transactions use/unlock the same input,*

$$\forall \tau \in \mathcal{T} \text{ and } \forall (x, l) \in I_\tau, \ \nexists\, \tilde{\tau} \in \mathcal{T} \text{ and } \tilde{\tau} \neq \tau \text{ such that } (x, l) \in I_{\tilde{\tau}} \tag{C2}$$

**C3**. **proof of ownership** – *for every input $(x^j, l^j) \in I_\tau$ the provided corresponding unlocking script $u^j \in U_\tau$ must cryptographically satisfy the input locking script $l^j$,*[9]

$$u^j \mid = l^j \tag{C3}$$

**C4. input-output funds balance** – *if a transaction's inputs and outputs are digital funds (e.g., BTC, ETH), then the total value of output funds $\sum_k y_\tau^k$ cannot exceed the total value of input*

[9]The locking script $l^j$ could be multisig, in which case two or more private keys may be required to satisfy (C3).

*funds* $\sum_j x_\tau^j$,[10]

$$\sum_j x_\tau^j \geq \sum_k y_\tau^k \tag{C4}$$

**C5. timelock enforcement** – *a timelocked input* $(x, l) \in I_\tau$*, where the locking script* $l$ *contains a timelock* $T$ *set by a previous transaction* $\tau'$ *with timestamp* $t_{\tau'}$ *cannot be used and locked into output* $(y, l') \in O_\tau$ *of a valid transaction* $\tau$ *before the timelock period* $T$ *has elapsed,*

$$t_\tau \geq t_{\tau'} + T. \tag{C5}$$

The algorithmic constraints C1–C5 ensure that a valid transaction cannot use inputs that are not available and/or not unlockable at the time of execution. A transaction is thus valid if it results, or would result, in successful unlocking of its locked inputs and transforming them into outputs secured by locking scripts. A transaction would be invalid if executing it results or would result in an error. Possible reasons can be that the input funds have already been unlocked and used by another transaction (C1 and C2),[11] the supplied unlocking script(s) do(es) not match the locking script(s) for all referenced inputs (C3), or if a coding or other error in the transaction results in violation of any of constraints C1–C5.

Importantly, a transaction $\tau$ can be invalidated deliberately by posting another transaction $\tau'$ that unlocks and uses $\tau$'s inputs so they would be unavailable if $\tau$ were submitted for execution. Clearly, an otherwise valid-as-written transaction $\tau$ can be rendered invalid in this way only prior to being posted on the platform, i.e., by executing $\tau'$ before $\tau$. In Section 2.3 I show how this mechanism can be used to implement secure payments between users.

### 2.1.4 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.

– *posting a transaction* – the act of submitting the transaction for execution on the digital platform.

I abstract from waiting time and assume for simplicity that all posted transactions are confirmed

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

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

and recorded without delay.[12] Once executed the transaction is irreversible.[13] 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.

Finally, I assume that an interface exists as part of the platform through which users can pass created transactions and scripts securely to each other and users can securely hold created transactions before or without being submitted for execution on the platform.

## 2.2 Algorithmic incentive compatibility and enforcement

In the following analysis I only focus on algorithmic (code-based) tools to implement payments and digital transfers. External or third-party commitment and enforcement technologies and institutions are assumed away. That is, I require that any stipulated payment or transfer must be secured and implementable by the platform code itself, i.e., self-enforcing, and no user should have an incentive to renege. The following definitions formalize these requirements.

**Definition 3 (Incentive compatibility in algorithmic contracts)** *An algorithmic contract or sequence of transactions is incentive-compatible if, at each possible node and history, executing the contractually intended action is best response for all agents. Formally, given the algorithmic constraints C1–C5 and the state of locked inputs, no agent can achieve a strictly higher expected payoff by deviating to an off-path action (e.g., reneging, withholding a signature, or misreporting a state).*

**Definition 4 (Self-enforcing algorithmic contracts)** *An algorithmic contract or sequence of transactions is self-enforcing if its incentive compatibility is sustained exclusively through the platform's algorithmic rules, including C1–C5 and the endogenous posting of digital escrow, without relying on external third-party enforcement. Formally, a contract is self-enforcing if the prescribed strategy profile is Subgame Perfect Equilibrium of the extensive form game defined by the contract.*

An implication of Definitions 2 and 4 is that for a transaction $\tau$ specifying a payment from $A$ to $B$ to be valid, it is essential that that the locked input(s), e.g., $(x, l) \in I_\tau$, exist and have not been used by any previous valid transaction (algorithmic constraints C1 and C2) and that a matching unlocking script(s) $u \in U_\tau$ with $u \mid = l$ is supplied when the payment is due, including satisfying any timelock or multisig script conditions present in the locking script $l$ (constraints C3–C5). In

[12] In actual blockchain platforms there are submitted transactions that may remain unconfirmed for some time, e.g., Bitcoin's *mempool*. I abstract from this and use the term "posted" to mean transactions that are both submitted and confirmed.

[13] A transaction can be constructed to "reverse" the ownership of digital funds or assets but such transaction would have different input and output sets and would be recorded as a new transaction.

practice, this means that the transaction input(s) must be secured/locked in escrow until the time of actual payment or settlement.

Securing the input funds is achieved through the algorithmic tools described in Section 2.1 – multisig, timelock and combinations thereof, and by executing a special collateral transaction (see next section) which activates these tools and secures the future availability of the inputs. Specifically, multisig locking scripts, e.g., $l = k^a \wedge k^b$ ensure that no single party can spend or divert the funds without the other party's agreement. Therefore, the input funds can be transferred (e.g., user A pays B) only when the economic incentives of both parties are aligned, ensuring mutual trust. Timelock scripts lock funds from use by any party and ensure that the inputs are still available at a later date. Applications and examples are discussed in the next sections.

## 2.3 Payment channels

Moving on to implementation and applications, I first show how the digital algorithmic tools from Section 2.1 can be used to implement a *payment channel* (Decker and Wattenhofer 2015), an incentive-compatible and self-enforcing algorithmic implementation of payments between two users.[14] An extension to multiple users is briefly discussed in Appendix B.

Consider a contract for payment between two agents, $A$ and $B$. To fix ideas, think of the agents starting at some 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 state transition is 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

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

where the value $c$ determines the payment channel capacity.

[14] For concreteness I follow the Bitcoin implementation of payment channels, consistent with the notation developed in Section 2.1, however, the specific implementation is not essential for the main results and conclusions.

### 2.3.1 One-way payment channel

To demonstrate the use of the algorithmic tools from Section 2.1 to implement payments on a digital platform, I first consider 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 a video streaming service. There are two main contractual frictions 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 illustration).

**o1. collateral/escrow set-up**. The parties create (this can be a functionality of the platform) a collateral transaction $\tau_c$ which establishes the maximum total payment that can be implemented via the payment channel, with elements,

$$I_{\tau_c} = \{(x_0, l_0)\}, U_{\tau_c} = u_0, O_{\tau_c} = \{(c, l_c)\} \tag{CT}$$

The input funds $x_0 = c$ of $\tau_c$ are provided by user $A$. 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 $c$ ($= x_0$) 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 $c$ in a transaction without the other party's agreement (private key).

**o2. refund set-up.** Before posting the collateral transaction $\tau_c$ on the platform, the parties also create a second, refund transaction $\tau_r$ with elements

$$I_{\tau_r} = \{(c, l_c)\}, U_{\tau_r} = u_c, O_{\tau_r} = \{(c, l_r)\}$$

in the output of which the escrow funds $c$ are secured by timelock $T > 0$ defined relative to the timestamp of the collateral transaction $t_{\tau_c}$ (e.g., one year in the future) using the locking script,

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

This transaction establishes the maximum duration of the payment channel. User $A$ passes $\tau_r$ to $B$ to sign, i.e., supply the $k^b$ part of the required unlocking script $u_c$. User $B$ agrees to sign, since

by assumption s/he stands to benefit from the underlying trade. 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 the escrow funds $c$.

**o3. posting collateral/escrow.** After $B$ signs the refund transaction $\tau_r$, $A$ posts the escrow transaction $\tau_c$ on the platform (setting timestamp $t_{\tau_c} = t_0$) 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 using the channel.

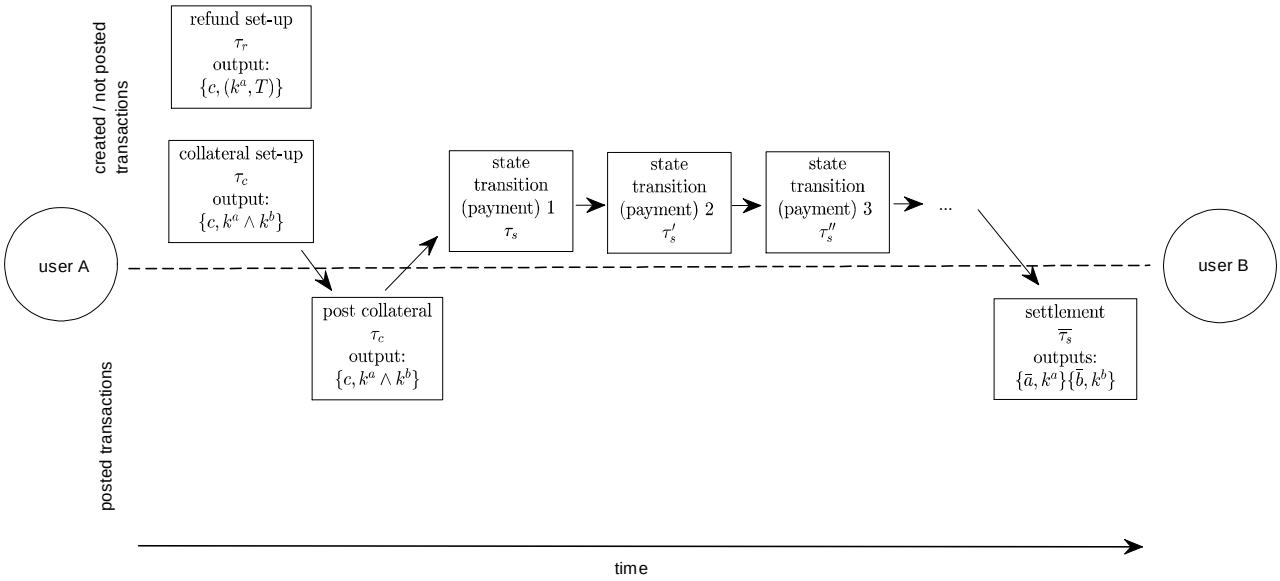


Figure 1: Payment Channel – Illustration

**o4. 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, $b' > b$, that is, $A$ is making a digital payment (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 inputs the escrow funds $c$ locked by $l_c = k^a \wedge k^b$ and two outputs corresponding respectively to the new balances of $A$ and $B$,

$$I_{\tau_s} = \{(c, k^a \wedge k^b)\}, U_{\tau_s} = u_c, O_{\tau_s} = \{(a', k^a), (b', k^b)\}$$

where $a' + b' = c$. User $A$ signs $\tau_s$ by his private key $k^a$ supplied as part of the unlocking script $u_c$ and passes it to $B$. User $B$ can then post $\tau_s$ for execution by signing it with $k^b$, which would complete the required unlocking script $u_c$ and transfer (give control of) $b'$ to $B$ and $a'$ to $A$. Alternatively, $B$ could hold on to $\tau_s$ if expecting additional payments from $A$ and hence a larger eventual balance. Further state transitions, that is, new payments from $A$ to $B$, are made in the same way – by creating new transactions, $\tau_{s'}$, $\tau_{s''}$ etc. Each of these transactions has as input the locked escrow $(c, l_c)$ and two outputs corresponding to the users' new balances, e.g., $a''$ and $b'' = c - a''$ with $b'' > b'$ and so on.

**o5. settlement.** The final state transition transaction, $\bar{\tau}_s$ (called settlement) must be posted before the expiry of the timelock $T$ set in step o2, i.e., it must have timestamp $t_{\bar{\tau}_s} < t_{\tau_c} + T$. Executing the settlement transaction $\bar{\tau}_s$ closes the payment channel since it unlocks the escrow funds $c$ and transfers them, via its outputs $O_{\bar{\tau}_s} = \{(\bar{a}, k^a), (\bar{b}, k^b)\}$, into the parties' final balances $\bar{a}$ and $\bar{b}$ with $\bar{a} + \bar{b} = c$. A new payment channel can be established by going back to step o1, e.g., if $A$ subscribes to the video stream again.

Observe that while many payments from $A$ to $B$ may occur over the duration of the channel, only two transactions are posted on the platform (and would incur transaction fees if applicable) – the collateral transaction $\tau_c$ and the settlement transaction $\bar{\tau}_s$.

**One-way payment channel – game setting and actions**

The algorithmic implementation of one-way payment channel transactions described above can be analyzed formally as a discrete-time game between agent $A$ (the buyer) and agent $B$ (the seller). Let $c > 0$ be the channel capacity established by the collateral transaction $\tau_c$. Assume that mutually beneficial gains exist so that the channel is established (steps o1–o3).

At any period $t$, the state of the payment channel is defined by the latest state transition (payment) transaction $\tau_{s,t}$ which allocates balances $(a_t, b_t)$ with $a_t + b_t = c$. In a one-way channel, $B$'s balance is strictly increasing over time: $b_1 < b_2 < \cdots < b_t$. Suppose that in each period $t$, user $A$ receives flow utility $f_t$ from the service provided by $B$ at marginal cost $p_t = b_t - b_{t-1}$. The action space at time $t$, when the last signed transaction is $\tau_{s,t}$, is:

User $A$'s possible actions:

– A1. sign the next state transition (payment) $\tau_{s,t+1}$

– A2. not sign the next payment transaction

– A3. post the refund transaction $\tau_r$ with inputs locked until $T$ expires.

User $B$'s possible actions:

– B1. provide the service

– B2. post an old signed transaction $\tau_{s,k}$ with $k < t$

– B3. post the most recent transaction $\tau_{s,t}$ to settle the payment channel and halt service

– B4. halt service without posting a settlement transaction.

Note first that $A$ cannot unilaterally post an old payment transaction $\tau_{s,k}$ with $k < t$ giving $A$ a higher balance state because transaction $\tau_{s,k}$'s input is locked by the multisig script $l_c = k^a \wedge k^b$ and $A$ does not have $B$'s private key $k^b$.

**Lemma 1:** *In a one-way payment channel, the seller's (agent B) optimal strategy upon settlement is to post the most recent payment transaction $\tau_{s,t}$.*

**Proof:** At time $t$ agent $B$ holds a set of valid counter-signed transactions $\{\tau_{s,k}\}_{k=1}^{t}$. $B$'s payoff from posting $\tau_{s,k}$ is $b_k$. By the definition of one-way payment channel, the payoff $b_k$ is strictly increasing in $k$. Therefore, posting any transaction $\tau_{s,k}$ with $k < t$ at settlement is strictly dominated by posting the most recent valid transaction $\tau_{s,t}$. ■

**Proposition 1: Subgame Perfect Equilibrium in one-way payment channel**

*The strategy profile where user $A$ signs the next payment transaction $\tau_{s,t+1}$ as long as her participation constraint is satisfied and posts the refund transaction $\tau_r$ if B halts service without settlement and user $B$ provides the service and posts the latest $\tau_s$ at settlement is a Subgame Perfect Equilibrium.*

Proof: see Appendix A.

**Discussion**

Proposition 1 shows how all payments performed in the one-way payment channel are made incentive-compatible and self-enforcing using the algorithmic tools from Section 2.1. Specifically, the use of the multisig locking script $l_c = k^a \wedge k^b$ in the collateral and payment transactions $\tau_c$ and $\tau_s$ ensures that neither party can unilaterally unlock and use the escrow funds. This protects $B$ (the seller in the example), by preventing the buyer $A$ from posting an old balance state (old $\tau_{s,k}$) that gives $A$ a larger balance than the most recent state. $A$ is prevented from posting such transaction by not having $B$'s cryptographic key to unlock the input funds – algorithmic enforcement.

Additionally, the timelock locking script $l_r = (k^a, T)$ used in the refund transaction $\tau_r$ protects $A$ by ensuring that s/he can eventually recoup her escrow funds $c$ if $B$ unilaterally quits and does not post a settlement transaction (B4). The timelock script also protects $B$ in case $A$ unilaterally quits / refuses to pay (A2) or posts $\tau_r$ (A3) by giving $B$ sufficient time to post as settlement the most recent payment transaction $\tau_s$ that $B$ holds.

### 2.3.2 Bilateral payment channel

The one-way payment channel analyzed in the previous section is only incentive-compatible and self-enforcing when the same party, e.g. $A$, is paying the other party ($B$), that is, more and more of the escrow funds $c$ become due to $B$ over time. To see this, suppose that, at some point along the payment chain, the parties wanted to perform a state transition in the opposite direction, i.e., $(a, b) \rightarrow (a', b')$ with $b > b'$, that is, $B$ making a payment $b - b' > 0$ to $A$. In such case the payoff monotonicity on which Lemma 1 and Proposition 1 rely would fail, since $B$ would gain from posting the old transaction $\tau_{s,k}$ with $k < t$ (already signed by $A$) giving $B$ control of the larger previous balance $b$.

Therefore, to enable bidirectional payments and rule out such deviations, a different implementation using the algorithmic tools in Section 2.1 is required. Formally, a bilateral payment channel supporting state transitions $(a, b) \rightarrow (a', b')$ with $a + b = a' + b'$ and either $b > b'$ or $b < b'$ (e.g., $A$ pays $B$ or $B$ pays $A$) is implemented as follows:

**b1. collateral/escrow set-up.** The parties create and post on the platform a collateral transaction $\tilde{\tau}_c$ with elements

$$I_{\tilde{\tau}_c} = \{(a_0, k^a), (b_0, k^b)\},\ U_{\tilde{\tau}_c} = u_c,\ O_{\tilde{\tau}_c} = \{(c, l_c)\}$$

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

**b2. counter-signed payment transactions and revocation scripts.** To perform a transition from balance state $(a, b)$ to a new state $(a', b')$ with $a + b = a' + b' = c$, where either $b' > b$ or $b' < b$ are possible, users $A$ and $B$ create and exchange two transactions, $\tau_{sa}$ and $\tau_{sb}$, held respectively by $A$ and $B$ and counter-signed by the other party's private key. That is, $\tau_{sa}$ is (pre-)signed by $k^b$ (indicated in (SA) as part of the unlocking script) and $\tau_{sb}$ is (pre-)signed by $k^a$, where:

$$I_{\tau_{sa}} = \{(c, k^a \wedge k^b)\},\ U_{\tau_{sa}} = \{k^b\}, O_{\tau_{sa}} = \{(a', (k^a, T) \vee r^b), (b', k^b)\} \quad \text{(SA)}$$

and

$$I_{\tau_{sb}} = \{(c, k^a \wedge k^b)\},\ U_{\tau_{sb}} = \{k^a\}, O_{\tau_{sb}} = \{(a', k^a), (b', (k^b, T) \vee r^a)\} \quad \text{(SB)}$$

These counter-signed 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.

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 as explained in step b3 below. Transactions $\tau_{sa}$ and $\tau_{sb}$ serve two roles: (i) implement the intended state transition (payment) and (ii) provide a refund guarantee, implemented via the timelock $T$.

**b3. state transitions (payments).** Transactions $\tau_{sa}$ and $\tau_{sb}$ defined in (SA) and (SB) implement payments as follows. Transaction $\tau_{sa}$ held by $A$ has the escrow $c$ as its input and is pre-signed with $B$'s key $k^b$, as indicated in $U_{\tau_{sa}}$. Hence, if $A$ signs $\tau_{sa}$ with $k^a$ and posts it for execution, the escrow funds $c$ would be unlocked and transformed into the two outputs $O_{\tau_{sa}}$, see (SA). Output $(b', k^b)$ immediately puts $B$'s current balance $b'$ into $B$'s control, unlockable by her private 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 "or" operator $\vee$, the funds $a'$ can be unlocked immediately by using the revocation 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 revocation scripts $r^a$ and $r^b$ and 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 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 – see the proofs of Lemma 2 and Proposition 2. The script exchange and counter-signing is performed algorithmically and simultaneously via the platform to avoid hold up. Final settlement is done by posting the latest transactions $\tau_{sa}$ and $\tau_{sb}$ before the timelock expiry.

**Bilateral payment channel – game setting and actions**

The bilateral payment channel transactions described above can be modeled a discrete-time game. The payment channel's capacity is $c = a_0 + b_0$. At any period $t$, the contractually agreed state is a balance distribution $(a_t, b_t)$ with $a_t + b_t = c$. Unlike in an one-way channel, user balances can increase or decrease in either direction. To implement channel state $(a_t, b_t)$ at time $t$, the agents create and hold a pair of valid counter-signed transactions $\tau_{sa,t}$ and $\tau_{sb,t}$. User $A$ holds $\tau_{sa,t}$ and a set of revocation scripts $\{r_k^a\}_{k=1}^{t-1}$ corresponding to all previous balance states. Similarly, user $B$ holds $\tau_{sb,t}$ and revocation scripts $\{r_k^b\}_{k=1}^{t-1}$.

The action space for user $i \in \{A, B\}$ at time $t$ is as follows:

– S1. agree to the next state transition by exchanging revocation scripts $r_t^a, r_t^b$ for the

current state and signing the next payment transactions $\tau_{sa,t+1}$ and $\tau_{sb,t+1}$

– S2. post the most recent transaction $\tau_{si,t}$ and stop

– S3. post an old transaction $\tau_{si,k}$ with $k < t$

– S4. execute revocation script $r_k^i$ with $k < t$

– S5. do not sign the next transaction or exchange $r_t^i$ or otherwise halt trade.

**Lemma 2:** *In a bilateral payment channel, posting an old transaction $\tau_{si,k}$ with $k < t$ is dominated by posting the most recent transaction $\tau_{si,t}$.*

**Proof:** Without loss of generality, suppose $A$ considers posting an old transaction $\tau_{sa,k}$ with $k < t$ (action S3) which gives $A$ a larger balance $a_k > a_t$. Note that this implies $b_k < b_t$ – i.e., a smaller than the current due balance for $B$. By the definition of $\tau_{sa}$ in (SA), posting $\tau_{sa,k}$ immediately allocates $b_k$ to user $B$, while $A$'s output $a_k$ is locked by the script $(k^a, T) \vee r_k^b$. Because $k < t$, user $A$ has already given revocation script $r_k^b$ to $B$ (action S1) during the time $k$ state update. Upon observing $\tau_{sa,k}$, user $B$ has a time window of length up to $T > 0$ to supply the revocation script $r_k^b$. Because $b_t > b_k$, it is in fact optimal for user $B$ to execute $r_k^b$ (action S4) and claim total payoff $b_k + a_k = c$. User $A$'s resulting payoff is $0$. Therefore, since the current channel state guarantees $A$ balance $a_t \geq 0$, posting an old transaction with $k < t$ is a dominated strategy, strictly if $a_t > 0$. ■

Lemma 2 proves that, for each user, posting an old transaction is dominated by posting the most recent transaction, since otherwise the other party can claim the full escrow value using the revocation script in their possession.

**Proposition 2: Subgame Perfect Equilibrium in a bilateral payment channel**

*The strategy profile where both agents update the channel state by exchanging payment transactions and revocation scripts, post the most recent transaction pair $\tau_{sa,t}, \tau_{sb,t}$ upon settlement, and execute a revocation script upon partner defection, is a Subgame Perfect Equilibrium.*

Proof: see Appendix A.

Because of the algorithmically-enabled penalties secured by the digital escrow, neither party has an incentive to post an old transaction. Specifically, the exchanged revocation scripts and timelocks ensure that, assuming the economic participation (gains from trade) conditions remain satisfied, neither agent can achieve a higher payoff by deviating. The bilateral payment channel is thus incentive-compatible and self-enforcing.

# 3 Applications

I next present two economic applications: a loan contract and a multiperiod digital insurance platform. I discuss implementation using the algorithmic tools from Section 2, specifically addressing the relevant enforcement or information frictions in these financial 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 and must repay $(1+r)l$ at the end of the loan term. I maintain the assumption of no external enforcement, that is, the digital loan contract must be implementable solely using the algorithmic tools from Section 2.

The main contractual friction in this loan example is enforcement. A 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 digital loan contract, each of $A$ and $B$ uses two platform accounts: an expenditure account, with balances $x^a$ and $x^b$ respectively, and an 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 corresponding agent $i = A$ 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 $a_0$ 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. The second inequality ensures that the loan is feasible. The escrow funds are provided via a collateral transaction and secured by the multisig script $m^{ab} \equiv k^a \wedge k^b$ and a timelock, as described below.

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

$$[x^a, c^a; x^b, c^b] = [0, a_0; 0, l]$$

where $A$'s escrow $c^a = a_0$ is secured with the script $\mathbf{s} = (m^{ab}, T)$ with timelock $T$ synchronized with the loan term/maturity.

Taking out the loan is implemented via a disbursement transaction $\tau_d$ in which amount $l$ (the transaction's input) is unlocked and debited from $B$'s escrow account balance $c^b = l$ and transferred into $A$'s expenditure account (the transaction's output) via supplying the unlocking script $m^{ab}$, i.e., the parties provide their private keys $k^a$ and $k^b$. The loan funds $l$ are thereafter redeemable (unlockable) by $A$ via her private key $k^a$ which results in new account balances

$$[l, a_0; 0, 0]$$

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

Loan repayment is implemented via posting a time-delayed (using a timelock) settlement transaction $\bar{\tau}_s$ backed by the escrow $a_0$ which debits the due amount $(1+r)l$ from $A$'s combined digital balance $x^a + c^a$ at the loan maturity date, prioritizing the expenditure account funds $x^a$ if available. 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 repayment is due and thus, when the timelock $T$ expires, the settlement transaction $\bar{\tau}_s$ unlocks and transfers the due amount $(1+r)l$ into the lender's expenditure account resulting in balances

$$[0, a_0; (1+r)l, 0]$$

At this time, the collateral $a_0$ is also unlocked by the timelock $T$'s expiry and becomes available to $A$, for instance to secure a new loan. In sum, by repaying as agreed, the borrower $A$ recovers her collateral funds 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 $T$ (defaults), the time-delayed settlement transaction $\bar{\tau}_s$ backed by the escrow $a_0$ results in the following account balances for the 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 a digital platform enabled loan is MakerDAO's borrowing facility implemented on the Ethereum blockchain. A user locks Ethereum cryptocurrency (ETH) as collateral and receives a digital token (DAI) denominated loan at a minimum $1.5$-to-$1$ collateral to loan ratio, for example, $\$150$ worth of ETH collateral maps into $\$100$ worth of DAI loan. 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 (lender).

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

### 3.2.1 Setting

A continuum of ex-ante identical risk-averse agents $i$ with preferences over consumption $u(c)$ and discount factor $\beta \in (0,1)$ have stochastic income $y_t^i$ for $t = 1, 2, ...$ where $y_t$ is a discrete random variable taking values $\{y_s\}$, $s = 1, ..., S$ with respective probabilities $\pi_s$ where $\pi_s > 0$ and $\sum_{s=1}^{S} \pi_s = 1$. The incomes $y_t^i$ are assumed i.i.d. over time and across agents. The agents have access to a risk-neutral digital insurance platform. The income realizations are private information.[15]

Using standard arguments, the optimal multi-period contracting problem in this setting can be written recursively as a dynamic program using 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 insurance platform requires each user to self-report their income. The objective 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 payout (indemnity), where the transfer amount $\tau_t^i$ received by user $i$ at time $t$ given income report $y_t^i$ and current state (experience/credit score) $w_t^i$ is

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

2. *a score updating rule,* $\phi(y, w)$ for the state variable $w$, i.e., for user $i$

$$w_{t+1}^i = \phi(y_t^i, w_t^i) \tag{3}$$

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

The timing within each time period $t$ is as follows:

(i) agent's income $y_t^i$ is realized;

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

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

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

(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 contracting friction is private information. An extension allowing for limited commitment (imperfect enforcement) is presented later in this section. The constrained optimal risk-sharing contract for a user with current promised utility $w$ and income $y_s$ solves the following dynamic programming problem:

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

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

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

In the above problem, the value function $\Pi(w)$ is the platform's profit 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 and provides actuarially fair (though 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 thus allows the optimal transfers $\{\tau_t^i\}_{t=1}^T$ to depend on 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 agent having constant consumption $c_s = \bar{c}$ across all income states $s = 1, ...S$, by receiving or making transfer $\tau_s^{fb} = \bar{c} - y_s$. This 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 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 level $\tilde{y}_s \neq y_s$ 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 via a digital platform using the algorithmic tools in Section 2 as follows. First, an offline computational module (based on or off the platform) is used to solve the dynamic programming problem and determine the optimal risk-sharing transfers (contributions or payouts) for any history of income reports given the (calibrated or estimated) income process $(y_s, \pi_s)$ and all other model parameters, e.g., for the user's preferences. The same module also computes the user credit/experience score (promised utility) updating rule $\phi$. The platform collects and records all income reports and scores data, and implements the optimal risk-sharing transfers (insurance contributions/premia or payouts/indemnities) for all users depending on their income histories.

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. In addition, each user is assigned a virtual token account (with no monetary value) in which the credit/experience score value $w$ (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 that the risk-sharing platform breaks even in expectation, over the long-term, with each user and hence overall too.

In each period $t \geq 0$, user $i$ with current score $w_t^i$ makes an income report $y_t^i$. The report is fed into the pre-computed 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 balance state transitions as described in Section 2.3. Specifically, the user and the platform start with balance state $(0, 0)$ (each has zero due) and initial user promised utility / score $w_0$ recorded in the user's digital token account. Then, given a history of user income reports $(y_{j_1}, y_{j_2}, ...)$ with $j_1, j_2, .. \in \{1, ..., S\}$ resulting in transfers $(\tau_{j_1}, \tau_{j_2}, ...)$, the following

chain of state transitions (transfers) is implemented via a bilateral payment channel:

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

where $\tau_{j_t}$ denotes the optimal 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 2.3, to implement such bilateral payment channel, an escrow amount $c$ needs to be set up spanning the maximum possible balance owed to either side. This escrow amount is secured by the algorithmic tools from Section 2.1. It is sufficient to set

$$c = \max\left(\frac{|\tau_{\min}|}{1-\beta}, \frac{|\tau_{\max}|}{1-\beta}\right)$$

where $\tau_{\min}$ is the largest possible transfer (contribution) from the agent to the platform and $\tau_{\max}$ is the largest possible transfer from the platform to the agent in the mechanism-design solution. If a user is due an insurance payout (i.e., $\tau > 0$), the amount is credited to the user's expenditure account while premia ($\tau < 0$) are backed by and drawn from the escrow account.

### 3.2.3 Limited commitment

In the risk-sharing setting with private information considered above I assumed that there is no enforcement problem on the user side, or alternatively, that the payment channel implementation from Section 2.3 ensures self-enforcement via the algorithmic tools. This is conditional on the users' ability to post the required escrow amount $c$ which, however, may be large.

Alternatively, the platform code itself can be adapted, using mechanism-design theory, to ensure that a user has no incentive to renege on a due transfer. Suppose that, after observing their income, the user considers refusing 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 provide an incentive for the user to stay in the contract (not renege) for any history of income realizations, by threatening to exclude or 'blacklist' the user from the platform (in a permissioned platform) or to close the user's account.

To provide this participation incentive, it is critical to know the user's outside option if their platform account were to be closed. Depending on the setting, the 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 for the limited commitment setting is that a higher outside option reduces the possible gains

from risk-sharing, hence a platform design that incorporates a credible threat of exclusion upon reneging would improve efficiency.

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 time and after any history of income realizations.

Using (prepaid) escrow 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 effective insurance transfer on the l.h.s. can be always made the same as in the constrained-optimal contract without escrow, $\tau^{i*}$ by setting $\tilde{\tau}(y_t^i, w_t^i) - f = \tau^{i*}$, however, the incentive to renege in high-income states when a contribution is due, is reduced because of the lower outside option value in the r.h.s. enabled by the escrow $f$. An appropriately chosen prepayment $f$ can therefore achieve improved insurance absent user liquidity constraints.

### 3.2.4 Numerical example

I present a numerical example illustrating the solution of the dynamic programs in the previous section and the welfare gains from improved risk sharing relative to autarky. Suppose user income can take two values: $y_L = 1$ or $y_H = 2$, 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) by solving the infinite-horizon dynamic program with initial promised utility/score $w_0$ set to satisfy zero ex-ante expected profits, i.e., $\Pi(w_0) = 0$. I compute three different information/enforcement settings – *hidden income* (imposing constraint TT), *limited commitment* (imposing constraint LC), and *both hidden income and limited commitment* (imposing both TT and LC).

Figure 2 displays the optimal transfers $\tau$ and consumption $c$ (income plus transfer) for all six

possible histories or length one period (2 histories) or two periods (4 histories), starting at the initial condition $w_0$.[16] For example, history $L$ on Figure 2 means low income ($y_L$) in period $t = 1$ while history $LH$ means low income ($y_L$) in period $t = 1$ and high income ($y_H$) in period $t = 2$, etc. The mechanism-design solutions are compared to autarky (the user consumes their current income, $c_t^i = y_t^i$; denoted by the dotted red line on Figure 2) and 'full insurance' (equal consumption across all states of the world; denoted by dashed lines), for each history.

Figure 2 demonstrates how the access to insurance, which can be provided via a digital platform as shown above, significantly improves consumption smoothing over income states and over time relative to autarky, despite the information or/and commitment frictions. In addition, the Figure clearly illustrates the history-dependence of the optimal insurance transfers (contributions or payouts), i.e., the exact sequence of income shocks, both in terms of size and order, matters.

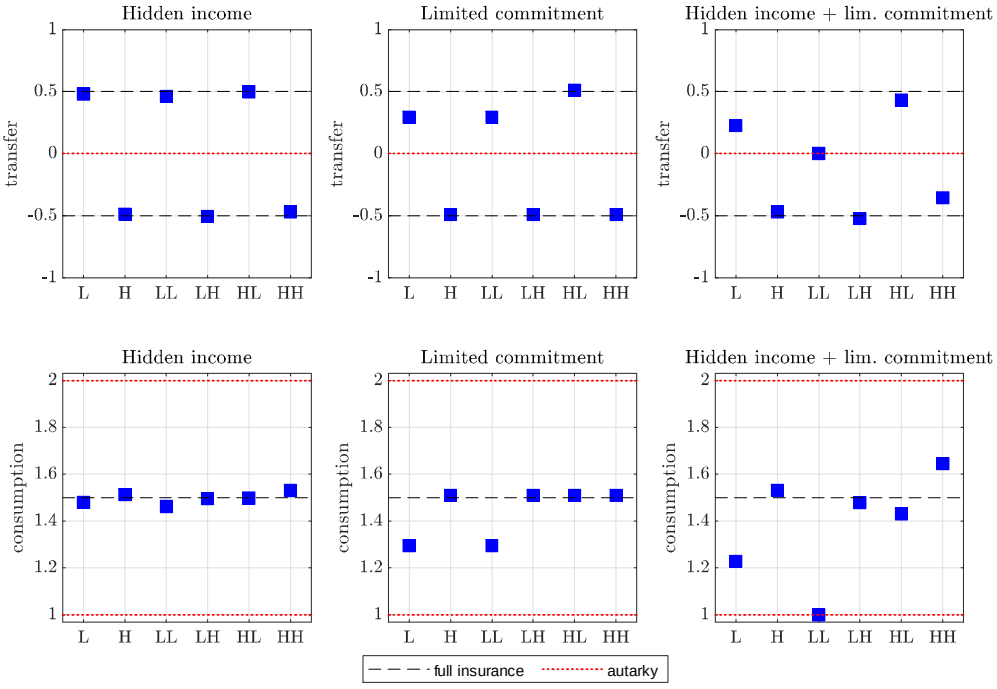


Figure 2: Optimal history-dependent transfers and consumption

Table 1 complements Figure 2 by quantifying the gains in consumption smoothing relative to autarky. Specifically, the table displays the (permanent) percentage consumption equivalent supplement in each state and time period that would make an agent who is in autarky indifferent to being in each of the alternative settings with improved risk-sharing implemented via the digital

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

platform.[17] We see that the welfare gains are sizable, including in the setting with both information and commitment frictions which registers the equivalent of $0.76\%$ higher consumption in each period and income state relative to autarky.

**Table 1. Consumption-equivalent gains relative to autarky**

| hidden income | limited commitment | both |
|---|---|---|
| $+0.97\%$ | $+0.99\%$ | $+0.76\%$ |

Implementation of the 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 income reports, in the 'hidden income' setting) which are then fed to smart contract code and mapped via the computed mechanism-design dynamic program solution to the optimal contingent transfer to or from the platform, depending on the present information or commitment/enforcement frictions. A digital token account is used to calculate and record an experience/credit score (mapped to promised utility) after each income report and the insurance transfers are function of the current income (report) and the promised utility token balance. Prepayment in escrow can be used to relax the limited commitment constraint as argued above.

# 4 Conclusions

Whether blockchains and other digital financial platforms are a transformative technology or conduits for financial speculation, fraud, or money laundering has been an ongoing debate. I contribute to this debate by modeling and applying key algorithmic primitives and capabilities of digital platforms in the construction and execution of financial transactions. I demonstrate that digital platforms based on mechanism-design theory offer significant potential for implementing complex financial contracts that can accommodate multi-period and history-dependent contingencies and conditionalities in economic settings characterized by private information and limited commitment.

A central challenge, especially in decentralized blockchain-based platforms, is the absence of external enforcement on which traditional financial markets and intermediaries rely. To address

[17] Specifically, Table 1 reports the percentage supplement $\xi^{alt}$ solving

$$\frac{Eu((1+\xi^{alt})c^{aut})}{1-\beta} = V^{alt}$$

where the l.h.s. is the lifetime present value of supplemented autarky consumption and $V^{alt}$ on the r.h.s. is the agent's present value in each of the three alternative financial settings starting from $w_0$.

this issue, I highlight the role of digital collateral (funds held in escrow), paired with algorithmic tools such as multisignature scripts and timelocks. These tools secure future payments by ensuring that funds remain available and inaccessible until predefined conditions are met, thus enabling trust and enforceability solely via computer code and without third-party intermediaries. I show how digital escrow can be used to back multiple one-directional or bidirectional payments through payment channels without posting transactions on the platform, thus saving on transaction costs.

One of the paper's main goals is to describe some of the technological innovations in digital platforms by breaking them down to a list of formally defined fundamental primitives and algorithmic tools. I provide a detailed analysis of platforms' specific functionalities and explore their applications in realistic economic contexts. A second goal is to advocate for the integration of economic theory, specifically mechanism design, into the development of digital platforms, which offers a structured approach to aligning user incentives, reports and actions with desired outcomes, in settings with 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 rating based on verified events or user reports. 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 or asset ownership. When combined with enforcement and trust mechanisms, either through platform-based escrow or, where appropriate, trusted third parties or mechanism-design solutions, these capabilities can unlock substantial gains. For example, a digital insurance platform can 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 or infeasible.

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

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

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

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

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

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

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

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

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

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

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

[13] 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

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

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

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

[17] 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

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

[19] 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

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

[21] 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*.

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

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

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

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

[26] Hau H. et al. (2024), "Fintech Credit and Entrepreneurial Growth", *Journal of Finance* 79(5): 3309-59

[27] He, D. et al. (2016), "Virtual Currencies and Beyond: Initial Considerations", *IMF Staff Discussion Note* 16-03.

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

[29] Hua X. and Y. Huang (2020), "Understanding China's Fintech Sector: Development, Impacts and Risks", *European Journal of Finance* 27: 321-33

[30] 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

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

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

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

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

[35] 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

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

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

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

[39] 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

# Appendix A. Proofs.

**Proof of Proposition 1**

Consider the agents' best responses and potential deviations at any time node $t$. Suppose $A$ chooses action A2 (does not sign the next transaction $\tau_{s,t+1}$), essentially stopping paying for the service. Then $B$'s best response is halting the service and, by Lemma 1, posting the most current valid transaction $\tau_{s,t}$ (action B3). Agent $A$ receives no further utility and has continuation value equal to the remaining escrow $a_t = c - b_t$ (i.e., no additional gain). If $A$'s participation constraint is satisfied, i.e., the current utility flow $f_t$ of obtaining the service exceeds the marginal cost of the state transition, $f_t > b_{t+1} - b_t$, then action A1 (signing the next transaction) strictly dominates A2.

Now suppose $A$ considers choosing action A3 instead, i.e., submitting the refund transaction $\tau_r$. Note that $A$ could do this unilaterally since $B$ already signed $\tau_r$ when it was created. However, executing $\tau_r$ and unlocking its input (the escrow $c$) requires the timelock period $T$ to expire. Because $T > 0$, $B$ has an ample time, e.g., from $t$ to $t_{\tau_c} + T$, to post the latest signed transaction $\tau_{s,t}$ that s/he holds and close the payment channel (action B3). By the algorithmic constraints C1 and C2, posting $\tau_{s,t}$ invalidates $\tau_r$ (uses its inputs) and therefore secures $b_t$ for user $B$. Therefore, $A$ cannot extract any extra payoff from this deviation beyond his current payoff $a_t = c - b_t$.

Next, I analyze $B$'s possible actions. By Lemma 1, $B$ has no incentive to post an old state transition i.e., action B2 is strictly dominated by B1 or B3. If $B$ unilaterally halts service and posts $\tau_{s,t}$ (action B3), then s/he secures payoff $b_t$ but cannot obtain any additional part of $A$'s balance $a_t$ because it remains locked by $k^a$ in $\tau_{s,t}$'s outputs. The only remaining possibility is action B4 (halting service without settling), however, then $A$'s best response is to post $\tau_r$ (action A3) and so $B$ gets nothing when the timelock $T$ expires and $\tau_r$ is executed. Hence action B4 is strictly dominated by B3.

Because the algorithmic constraints and specified transaction inputs, outputs and locking scripts ensure that neither agent can achieve a higher payoff by deviating at any node, the sequence of transactions in the one-way payment channel is incentive-compatible and self-enforcing. ■

**Proof of Proposition 2**

Let us analyze the agents' best responses and potential deviations at any time node $t$. The game is symmetric for both agents, so I focus on one of them. Lemma 2 shows that action S3 (posting an old state) is dominated by posting the most current state transition (S1 or S2).

Suppose an agent refused to update the state or cooperate (action S5). For example, suppose user $A$ refuses to exchange revocation scripts or sign the next state transition $t + 1$. Then user $B$'s best response is to post the latest transaction $\tau_{sb,t}$ (action S2). In such case, using (SB), $A$ receives $a_t$ and $B$ is guaranteed to receive $b_t$ when the timelock $T$ expires. Importantly, note that

by deviating user $A$ cannot gain more than $a_t$ since s/he does not possess the revocation script $r_t^a$ for the current, unrevoked state. That is, $A$ gains no additional payoff from halting cooperation (including agreeing to a settlement last transaction).

Finally, note that because of the algorithmic constraints C1-C5, $A$ can only choose action S4 (execute $r_k^a$ with $k < t$) if an old transaction $\tau_{sb,k}$ has been posted by the other user. Hence action S4 cannot be used as a unilateral deviation but only as punishment, as described in the proof of Lemma 2.

# Appendix B. Multi-party transactions

The basic idea behind bilateral payments described in Section 2.3 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 so-called 'hashed timelock script' using the hash function $H(\sigma)$ of a secret message $\sigma$.[18] Hash functions $H(\sigma)$ are mathematical functions that take as input a string of any finite length $\sigma$ and return a value $h = H(\sigma)$ of fixed length. Importantly, hash functions are easy to compute but practically impossible to invert, i.e., recover the message $\sigma$ when only knowing its hash value $h$. Formally, using the notation from Section 2, the hashed timelock script can be expressed as:

$$\boldsymbol{s} = h \vee (k^a, T)$$

Using the unlocking script $\boldsymbol{s}$ anyone with knowledge of $\sigma$ (and thus $h = H(\sigma)$), can unlock/use the locked input funds immediately. Alternatively, a user can unlock the inputs by providing the

[18] E.g., as implemented in Bitcoin's Hashed Timelock Contract (HTLC).

private key $k^a$ but only after the timelock period $T$ expires.

When payment is agreed upon, the recipient/payee $D$ creates the message $\sigma$ and sends its hash value $h$ (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

$$l_{\tau_1} = h \vee (k^C, T^1).$$

The payer $C$ then passes this transaction to the first intermediate node $N_1$ with whom $C$ is assumed to have a pre-established payment channel with capacity at least $p$. Note that user $N_1$ cannot redeem the locked funds since s/he does not know $\sigma$ or $H(\sigma)$ but can be incentivized by a small fee to pass along the locked funds to another user (e.g., user $N_2$ with whom $N_1$ has an established payment channel), with the output funds locked by the script $l_{\tau_2} = h \vee (k^{N_1}, T^2)$ with $T^2 < T^1$, and so on. The path $N_1, N_2, ...$ and locking scripts $l_{\tau_1}, l_{\tau_2}, ...$ can be constructed algorithmically by the platform code. The role of the timelocks $T^t$ 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 from some node $N_n$, since $D$ knows the secret message $\sigma$, she unlocks/claims the agreed payment $p$ from her bilateral payment channel balance with user $N_n$. Claiming $p$ algorithmically triggers sending the secret $\sigma$ to $N_n$ who in turn claims $p$ plus a small fee, $\frac{\varepsilon}{n}$ from his payment channel balance with user $N_{n-1}$ and thus benefits by $\frac{\varepsilon}{n}$. This process continues until we reach back $C$ where node $N_1$ claims $p + \varepsilon$ from $C$ via their bilateral payment channel. The end-result of this chain of 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 need to be posted on the platform, as long as there are pre-existing bilateral payment channels forming a path of platform users from $C$ to $D$.