Distributed Ledgers

Design and Regulation of Financial Infrastructure and Payment Systems

OA Funding Provided By:

The open access edition of this book was made possible by generous funding from Arcadia—a charitable fund of Lisbet Rausing and Peter Baldwin.

The title-level DOI for this work is:

doi:10.7551/mitpress/13382.001.0001

6

Smart Contracts: Contract Theory and Mechanism Design

The relevance of contracts comes to life and is supercharged with the possibilities created by distributed ledger technology, which allows "smart contracts." Likewise, we can be more precise in this section about underlying frictions and how smart contracts can deal with them. Finally, we return to the diverse perspectives of mechanism design and computer science and find some unexploited common ground that could be used in subsequent designs.

6.1 Smart Contracts

At their most basic level, cryptocurrencies such as Bitcoin use a language familiar to accountants, economists, and computer scientists. The state of the system is the current ownership of a digital asset, the *stock*, and a transaction or transition is the change in its ownership, the *flow*. These stocks and flows are on ledgers. Bitcoin and the other validation systems are all about verifying and validating flows back to the genesis state where assets were originally created. This ties flows to stocks and requires verification of information.

Usually, though, the concept of ledgers is generalized to mean simply lists of "facts." Below we will draw on the language of the smart contract composer Corda. Everyone has

a ledger, but it is synchronized and held in common only for shared facts. Consensus is broken into two pieces. Validity consensus means that a transition or transaction is accepted—that is, it has the required signatures, both for the current proposed transaction and for every transaction that led up to the proposed transaction. This is similar to the crypto-asset example. Unique consensus, in the language of Corda, is different and is the key to the generalization: A given party may not have a record of every single transaction, and that is not always required. There is not a consensus. On the other hand, a party could potentially request missing transaction information from notaries. The latter is necessary to thwart the double-spending problem, for example. The point here is that what is needed depends on the underlying environment and what the ledger is trying to accomplish, ideally as part of a constrained-optimal arrangement. Some transactions within a contract are private to the parties and there can be partitioning. Other transactions for value transfer may require public validation (more on the Corda notaries momentarily).

We have adopted in this section the language of Corda because it is perhaps the closest to the language of mechanism design, thus bringing computer science and economics together. That said, Ethereum is a well-known smart-contract protocol, close in its conception to Bitcoin, in which virtually all validations, even within a contract, are done by proof of work.

A contract is entered into by multiple parties. Parties are nodes. Identities of nodes could be anonymous, as in Bitcoin, but this is not required. Identities could be named and public—for example, the legal identity of an organization or the service identity of a network service. Note that trusted parties such as banks are allowed to be there as nodes, and to be public, but so too are others, as well as strangers. A node writing contracts is an app providing a service, which is also allowed to be a universal service, if desired. Technically, the smart contract itself

is a node, but it is acting autonomously as per its code (as is made clear in the subsequent paragraph). The permissioned set of nodes for a given contract in Corda still has its access controlled by a doorman, so in that sense all contracts are only semiprivate, and there is a sense of "centralization."

The contract is a code that is validated initially. It either works or not, and one can imagine several independent validators of code, dealing with potentially malfunctioning nodes, which also link back to the Byzantine Generals Problem (bad actors) and to thwarting potential collusion. Ethereum validates all lines of code by proof-of-work consensus, but that is not required in other smart-contract composers (we will return to this subject later).

A contract agreement is made via public and private keys. After these initial validation steps, it becomes immutable. In this sense there is no reneging on whether agreements have been entered into, nor claims that they are written in a different way. There is no need for trust on these particular dimensions. It is clear to outsiders what the parties intended. Smart contracts are stored and executed on a distributed ledger, an electronic record that is updated in real time and intended to be maintained on geographically dispersed servers or nodes. Through decentralization, evidence of the smart contract and its execution can be deployed to some or all nodes on a network, which effectively prevents modifications not authorized or agreed to by the parties.

A contract specifies states at a point in time (current ownership, for example) or other facts. Communication under a contract is node to node, not necessarily broadcast to the entire public, as in Bitcoin, but on a need-to-know basis, as prespecified in the contract. An *oracle*, a term computer scientists use to denote a function or node that knows the answer, is used to verify known facts that are states of a contract. Commands initiate transfers and result in output states.

Corda allows a notary service for validation of communication and proposed transactions within the contract. Upon receiving a proposed transaction, the notary will either accept it, if the notary has not already signed other conflicting transactions, or will reject it, as would happen in an attempt to double-spend. Every state has an appointed notary, and a notary will only notarize a transaction if it is the appointed notary of all the transaction's input states. A notary may be a single network node, in which case this part is quite centralized and has a trusted-third-party aspect. Alternatively, there can be a cluster of mutually trusting nodes to deal with faults or mutually distrusting nodes to deal with incentives. Though some Corda applications have multiple notaries, this is more for latency issues, and the incentive motivation for multiple notaries is not evident. We come back to a merger of the computer science and economic points of view at the end of this chapter. Notaries can choose a consensus algorithm based on privacy, scalability, legal system compatibility, and algorithmic agility. A notary could decide not to provide validity consensus, though in some contexts this runs the risk of denial of state attacks.

The key capabilities of smart contracts are that they overcome underlying frictions. Smart contracts allow full commitment, immutability, conditionality, observability, and enforceability. We come back to these ideas in the subsequent sections.

Nick Szabo, the inventor of smart contracts, states the advantage succinctly: Smart contracts would enable both parties to observe the other's performance of the contract, guarantee that only the details necessary for completion of the contract are revealed to both parties, and be self-enforcing to eliminate the time spent policing the contract (Gord 2016).

Agents in a contract use digital signatures: Private cryptographic keys are held by each party to verify participation and assent to agreed-on terms. A smart contract will take actions (e.g., disperse payments), without further action by the

counterparties, and they can access or refer to outside information or data to trigger actions.

Smart contracts fit naturally with elements of mechanism design; that is, they allow the execution of the kinds of contracts and mechanisms that economists have largely taken for granted: the revelation principle, enduring relationships, promised utilities, resolutions of the hold-up problem, and trusted reputation. We now elaborate on each of these in turn.

6.2 Mechanism Design

6.2.1 Messages

A game or mechanism among players is a specification of messages that can be sent, message spaces, and an allocation rule mapping realized messages onto outcomes such as transfers. Consider a Bayesian Nash equilibrium of such a mechanism, with strategies for play in which the strategy of any party to the contract is to maximize taking—that is, given the maximizing strategies of others, taking expectations relative to given information. The *revelation principle* asserts that for any given game with abstract message spaces and allocation rules, there is an alternative game in which messages can be reduced to statements about underlying unobserved facts and private information, and in this new, modified game, agents have incentives to tell the truth about privately observed states. This is without loss of generality (Harris and Townsend 1981; see also Dasgupta, Hammond, and Maskin 1979 and Myerson 1982). There are three interrelated points. The first point is that messages would be transmitted by actors or nodes and put on distributed ledgers. A second, related point is that the messages become endogenous objects. We have moved from how to design database and communication systems to the incentives that create the underlying data to be transmitted. Third, messages are truthful, because of underlying incentives, and so

Box 6.1
Key elements from mechanism design.

Single period contract with messages

There are two agents: #1 a villa as an agent, who observes a vector θ of its current state (e.g., its harvest output), and #2, the central estate as principal, who does not see it. Under this resource allocation scheme, villa 1 waits to see output vector θ before sending message m. Thus its decision problem is of the form, for every $\theta \in \Theta$, maximize $U^1[\theta - f(m)]$ by choice of $m \in M$. The agent sends a message m about the value of θ to the principal, choosing m from a set of possible messages M agreed to in the contract, and as a function of the message sent, is taxed, or receives a transfer if negative, $f(m)$, to the principal.

Multi-period contracts: histories and immutability

A two-period contract (with randomized rewards) occurs when agent 1 at date $t = 2$, given some history of announcements at $t = 1$, $\tilde{\theta}_1$, is given incentives to announce current actual θ_2 truthfully as opposed to any counterfactual value $\tilde{\theta}_2$. These messages result in lottery $\pi(\tau)$ over transfers τ, and expected utility is the metric payoff for agent 1:

$$\Sigma_\tau U^1[\theta_2 - \tau]\pi_2\left(\tau \mid \tilde{\theta}_1, \theta_2\right) \geq \Sigma_\tau U^1[\theta_2 - \tau]\pi_2\left(\tau \mid \tilde{\theta}_1, \tilde{\theta}_2\right). \qquad (1)$$

The history of announcements $\tilde{\theta}_1$ is public and immutable. Working the dynamic program backward to $t = 1$, agent 1 is given incentives to announce actual θ_1 truthfully, as distinct from some other $\tilde{\theta}_1$, taking into account that agent 1 will want to be truthful at date 2, as derived in equation (1).

The incentive constraints at date 1, for every actual θ_1 and announced $\tilde{\theta}_1$, are presented as

$$\begin{aligned}&\Sigma_\tau U^1[\theta_1 - \tau]\pi_1(\tau \mid \theta_1) + \beta\Sigma_{\theta_2} p(\theta_2 \mid \theta_1)\Sigma_\tau U^1[\theta_2 - \tau]\pi_2(\tau \mid \theta_1, \theta_2) \\ &\geq \Sigma_\tau U^1[\theta_1 - \tau]\pi_1(\tau \mid \tilde{\theta}_1) + \beta\Sigma_{\theta_2} p(\theta_2 \mid \theta_1)\Sigma_\tau U^1[\theta_2 - \tau]\pi_2(\tau \mid \tilde{\theta}_1, \theta_2).\end{aligned} \qquad (2)$$

In equation (2), $p(\theta_2|\theta_1)$ is the probability of θ_2 conditioned on θ_1.

do not need to be validated other than for the reliability of the computers that send the messages. This separates the external validation problem from the internal contract part, a hugely beneficial step.

As an example, consider an economy with one period, two agents, and multiple commodities or multiple financial assets. One agent's shocks to utility or its value function for profits determine the trade-off in the objects it wants to consume or assets held, but these shocks are not seen by the other party. The other party is, however, in this example, indifferent to the various combinations of what is consumed or held. The first agent with private information sends messages announcing the agent's preference shocks to the second party, messages that are part of the ledger. The outcome function specifies transfers of goods, hence requiring shipment, and/or transfers of securities, which can be done online in registries. The first agent has an internal incentive to send the correct, truthful message, because the agent bears all the consequences of its announcement.

6.2.2 Impact of Enduring Relationships (Duration): Past History of Messages Becomes Committed and Creates a New State as Part of the Determination of Contemporary Outcomes

Distributed ledgers housing smart contracts can help to implement formal and enduring relationships (Townsend 1982) that extend over time. The idea as indicated earlier is to give a household, firm, or trader an incentive to reveal private information. Sometimes we take this for granted. A household with low but privately observed income will have an incentive to implicitly announce or reveal low income by the act of requesting a loan, assuming here for the moment that loans must be paid back in the next period and that this is not an issue. Conditionality with collateral can ensure repayment, but there are

other means to ensure repayment, which we address below. To return to the main thread, the opposite outcome holds for households with high income today that voluntarily invest, with more funds coming back, available for the next period.

A bit more formally, we can imagine an economy with two agents, one borrowing/investing with variable income and a second who is willing to enter into the contract. The first agent has variable endowments over time and is risk averse, caring about ex ante expected utility. The second is risk neutral with essentially unlimited deterministic resources. Income realizations of the first agent are not public. The two agents agree and enter into a contract at some initial date, and it is then carried out over time. Though borrowing and lending form an incentive-compatible contract with private information, it is not the best that can be done. The optimal information-constrained arrangement is a blend of borrowing/lending and insurance. The constrained-optimal contract contains more risk contingencies in the contract relative to borrowing and lending, as the contract is entered into before the underlying state is known, allowing some insurance against the state. If income is low, for example, the borrowing agent acquires more money than from borrowing alone, as if an indemnity for low income is paid, and this agent faces a lower inter-temporal interest rate on the loan to be repaid. In turn, if income is high, the agent pays in an ex post premium, receiving some lesser amount of money back the next period. Longer contracts are better.[1]

A two-period or longer-term relationship contract is implemented using messages of unobserved states that are validated and put on a consensus ledger. For example, past messages of high- or low-income states both determine the outcome in the current period, pay out or receive, but also the state for the next period. That is, future pay receive, pay-in schedules are conditioned on current messages as well as messages in the next period.

6.2.3 Promised Utility as the State

To generalize, and thinking of longer-term relationships, the contract payoff can be divided into two pieces. One piece is a contemporary reward or penalty today, with goods, money, and/or tokens changing hands as a function of the message today. The other piece is the discounted expected utility of continued participation from the next period onward, varying with the outcome and action.[2] This expected utility is the key state variable to be entered on ledgers as a state variable for tomorrow triggered by states as messages today. Beginning-of-period promised utility is the state of the contract. The messages satisfy truth-telling or incentive constraints, so there is no need for trust other than presumed maximization. These utility numbers are under full control of the long-term contract, and here in this example they are public, with the conditional possibilities for future periods a function of such facts/states. The idea is to create incentives to announce states truthfully.

6.2.4 Incentives to Take Appropriate Action

An example that is explicit about unobserved actions taken in a contract is a multi-period, principal-agent insurance problem. The agent is a household or firm that enters into a contract with a risk-neutral insurer. The agent can take an action that makes loss less likely or profits higher, but this action is not seen by the insurer. If full insurance against loss or profit fluctuation were agreed to, the agent would have no incentive to take a costly action, hence the term *moral hazard*. This is indeed the classic moral-hazard problem (Harris and Raviv 1979). The constrained-optimal contract does not provide full insurance or zero insurance. Instead, it balances off the gains from insurance against this distortion in actions. If final losses or profits are fully observed, though again the action is not, the optimal contract culminates with a state-contingent transfer from the

Box 6.2
Dynamic principal and agent problem with moral hazard and promised utility.

The dynamic optimal contracting problem between a risk-neutral lender and the household is described as follows. The value function *V(w, k)* of the principal is the discounted expected present value of the dynamic contract from the current state, promised utility w to the agent and capital k of agent, choosing a mixture or lottery over current transfer τ to the agent, induced effort z on the part of the agent, observed output q from the firm run by the agent, an assignment of promised utility for the next period w', and investment, hence next period capital stock k'. The *V(w, k)* is the contemporary award to the principal, $q - \tau$, and pay off next period $V(w', k')$ discounted by the outside economy-wide gross interest rate R.

$$V(w,k) = \max_{\{\pi(\tau,q,\overline{z},k',w'|k,w)\}} \sum_{T\times Q\times Z\times K'\times W'} \pi(\tau,q,z,k',w'|k,w)$$
$$[q-\tau+\left(\frac{1}{R}\right)V(w',k')]$$

Consistency in promises requires as a constraint that previous promises resulting in current state w must be consistent with the utility earned by the agent with contemporary utility function U, receiving transfers while in control of the capital stock, partially depleted by deprecation rate δ and adjusted for capital carried over to the next period. Effort z enters into contemporary utility as a negative disutility term. The agent discounts the future at rate β, pre-multiplying next period promised utility w', a control variable of the contract.

Promise-keeping

$$\sum_{T\times Q\times Z\times K'\times W'} \pi(\tau,q,z,k',w'|k,w)\,[U(\tau+(1-\delta)k-k',z)+\beta w'] = w.$$

There are additional Bayes rule consistency, adding-up, and nonnegativity constraints.

Incentive compatibility is ensured by the moral-hazard constraint that the agent should take the action $\overline{z}$ recommended under the contract,

with utility on the left-hand side of the inequality, as distinct from any other action $\hat{z}$ resulting in utility on the right-hand side of the inequality, where if $\hat{z}$ were taken, the probability of output q given k has to be adjusted according to a likelihood ratio.

Moral hazard

Additional constraints are incentive compatible, $\forall(\overline{z}, \hat{z}) \in Z \times Z$.

$$\sum_{T\times Q\times K'\times W'} \pi(\tau, q, \overline{z}, k', w'|k, w)\,[U(\tau + (1-\delta)k - k', \overline{z}) + \beta w']$$
$$\geq \sum_{T\times Q\times K'\times W'} \pi(\tau, q, \overline{z}, k', w'|k, w)\,\frac{P(q|\hat{z}, k)}{P(q|\overline{z}, k)}\,[U(\tau + (1-\delta)k - k', \hat{z}) + \beta w']$$

insurer to the agent as a function of those observables. If losses or profits are unobserved, the agent must be given an incentive to tell the truth in messages as well, as discussed earlier. If the relationship does not terminate, then in addition to the current transfer, promised utility is updated for the next period. Essentially the agent takes into account rewards in terms of both higher current consumption (dividends) and higher future utility, and vice versa, with penalties for low states.

6.2.5 Utility Threats

Fernandes and Phelan (2000) generalize with unobserved past states to include yet another dimension: utility threats. There are upper bounds for what one can get if one has deviated in the past and contemplates lying or disobedience now. The threat bounds are high enough to keep agents on the equilibrium path so that there is no deviation in the first place.

If we are capturing the underlying economic environment accurately, these expected utility numbers or utility threats will indeed be the actual realized subjective utility an agent would experience, or get in the event of deviation, in that

environment under the incentive scheme. This is also a qualification. If the approximation of the model to the actual economic environment is poor, then these promised utilities and threats in the contract will be poor approximations of actual subjective utility.

6.2.6 Costly State Verification: Limited Message Spaces and No Need for Validation

As a third example on this theme, output from a project is privately observed but verification of project output by a lender or insurer is possible at a cost. In effect, this is a costly verification of the underlying situation that generated the messages and hence in principle could be part of computer science designs. Over a range of outputs, repayment of a loan is constant, resembling debt. Actual outcomes need not be known, and in effect there is no message and no need for validation. For low outputs, however, claims are verified at a cost, as if validating financial statements, for example. This is costly state verification (Townsend 1979; Gale and Hellwig 1985).

6.2.7 Various Meanings of the Word "Trust" in Economics, as Distinct from Limited Commitment

The general point here is that the word "trust" in mechanism design has various meanings—trust that an agent will announce unobserved states—but more likely this agent is given an incentive to do so, as a result of intra-temporal or inter-temporal considerations. Similarly, there could be trust that an agent will take appropriate actions, but more likely the agent is given an incentive to do so by reward schedules. In particular, incentives are separated from full commitment, the promise to do something regardless, as distinct concepts. Full commitment without incentives is akin to naïve communication protocols followed by trusted agents. To be fair, all the above examples are presumed to have some full commitment

on the part of the parties to carry out the agreement—that is, pay for goods transfers or repay a loan, without reneging.

To elaborate, the simplest of contracts between two parties is the purchase and sale of a commodity or asset. Indeed, if done as a spot-market exchange, then we can avoid the language of contracts altogether. But here it is clarifying to think of the contract as an agreement on the part of the buyer to surrender cash and the seller to surrender the commodity. More generally, the buyer is debited cash and the seller is credited. When there are lags between the time of the agreement to trade and the eventual payment, then issues can arise. If there were no collateral, then the parties have to trust each other to carry out their part, pay cash or ship goods. This is a full-commitment contract under which promises are made and honored, as is commonly assumed in the contract literature, hence, one notion of trust. However, an alternative is for a trusted third party to stand between the traders making trade possible among strangers. PayPal and Alibaba/Ant Financial are examples.

With borrowing and lending, trust issues are even more apparent, as the lender either trusts the borrower to repay or, otherwise, the collateral backing the loan should be placed in escrow. In the latter case, a transaction among strangers is possible, implemented with a conditional if-then statement to define what happens with and without default; see Geanakoplos (2003) and Kilenthong and Townsend (2018) for a literature on securities as contracts that embed collateral as a contract characteristic.

In contrast, under limited commitment without a third-party intermediary and without collateral, one party may wish to withdraw and go their own way as the contract unfolds. In this case contracts with limited commitment make sure that future rewards do not fall below certain thresholds, in order to retain participation. This is loaded into the underlying contract

itself. In the autarky version of this, there is implicit trust that a banishment penalty can be imposed were withdrawal to happen. Ironically, it takes this off-equilibrium commitment to deal with the original limited-commitment problem.[3]

For example, in a multiparty insurance arrangement, an agent with realized high income is supposed to pay an ex post premium, as a gift to others, into the insurance pool. But continued participation may yield lower interim expected utility for that agent than not paying and going it alone. These

Box 6.3
Limited commitment.

An additional constraint is appended to the equations for box 6.2. Let $v^{\text{aut}}(k')$ be the discounted utility from the next period on if the agent takes capital k' into autarky on their own. Let $\Omega(k, q, z)$ be the utility at the point of withdrawal, with current capital k given, effort z taken, and q realized, plus discounted $v^{\text{aut}}(k')$. The limited-commitment constraint ensures that with $\overline{q}$ and $\overline{z}$ given, if the agent stays in, expected utility as determined by transfers and promises for tomorrow and next period's capital is not less than the utility of withdrawal into autarky, defined above.

Additional constraints, limited commitment, for all $(\overline{q}, \overline{z}) \in Q \times Z$, are

$$\sum_{T \times K' \times W'} \pi(\tau, \overline{q}, \overline{z}, k', w' | k, w) [u(\tau + (1-\delta)k - k', \overline{z}) + \beta w'] \geq \Omega(k, \overline{q}, \overline{z}),$$

where $\Omega(k, q, z)$ is the present value of the agent going to autarky with the agent's current output at hand q and capital k, which is defined as

$$\Omega(k, \overline{q}, z) \equiv \max_{k' \in K'} \{u(\overline{q} + (1-\delta)k - k', \overline{z}) + \beta v^{\text{aut}}(k')\},$$

where $v^{\text{aut}}(k)$ is the autarky-forever value.

participation constraints, as limited-commitment constraints, are now part of the original contracting problem. In effect, when a bound is hit, the contract resets so that the party tempted to renege gets more out of the arrangement.

Likewise, in the competitive market version of this, we trust a party not to break the relationship even though it might be advantageous not only to one of the original parties to the contract but also to an entrant. Specifically, one party of a contract is drawn off by a competitor or a third party that has not entered into the original, multi-agent arrangement (Jacklin 1987). Relatedly, even within the arrangement, both parties to a contract may wish to renegotiate and start over, but that would be bad for incentives ex ante. So, either the parties trust that will not happen, as in full-commitment models, or a time-consistency constraint is appended to the underlying contract problem so that there is no temptation.[4] Enhanced commitment to deal with single-party or multiple-party deviations is possible under the smart-contract technology.

6.2.8 Reputation

Otherwise, if contracts are incomplete and penalties cannot be imposed internally as part of the contract, then there is a way to formalize a role for reputation when there is no trust. There is a way to design and commit to optimal social penalties through a scoring function (Lehnert, Ligon, and Townsend 1999). This kind of indirect scoring does improve commitment in the broader sense. The notion that we trust third parties because we know they are worried about their reputations is formalized in this literature. It is not taken for granted. It is modeled and is part of the design. Nodes can be given incentives to validate correctly, for example. With these in place we have trust, but a more nuanced version—in effect something like trust but verify, as in nuclear disarmament. Here, it is trust but it is implemented

through scoring. One can "trust" third parties without thinking all this through, but obviously things can go awry.

6.2.9 The Economics of Collusion and How to Prevent It

We have been discussing how to find constrained-efficient, ex ante agreements via maximization subject to truth-telling constraints, moral-hazard constraints, and limited-commitment constraints. However, though a solution exists and constitutes a valid (Bayesian) Nash equilibrium, typically the solution is not immune to collusion. That is, players need not act in the Nash sense of taking the strategies of others as given, but rather could all act in concert to enhance their welfare at the expense of the principal. One example is the discussion of Bitcoin miners discussed previously. Another example would be auctions where agents can collude in their bids. So either we trust parties not to collude or, as can now be anticipated, we place additional constraints on the multiparty contracts. The main point: These kinds of multiparty conditioning statements can be programmed under smart-contract technologies. The outcome is an explicit, contracted function of what each player says or does, in potentially complicated ways, so that deviations from a proposed collusion solution dominate (i.e., eliminate collusion).[5]

6.2.10 Implementation through Sequential Play: Loading in Commitment That Was Missing Previously

Holden and Malani (2018) examine how to use the blockchain mechanism to resolve the hold-up problem in economics: ex post bargaining when the contract is incomplete, without commitment. In their scheme, choices that cannot be verified by a third party can be resolved by a blockchain mechanism that can commit the parties to not engage in renegotiations. The keys here are the posting of agreements, having them notarized by multiple parties as witnesses, and the security and inalterability of the blockchain. Moore and Repullo (1988) and Maskin and

Tirole (1999) resolve an implementation problem using a simple sequential mechanism, as in Moore (1992): When one party tells the truth, that party earns more than if they lie. Joshua Gans (2018) shows how this implementation mechanism can be achieved with smart contracts to solve a trade problem.

6.2.11 Ledgers of the Financial Accounts

Contracts interface with ledgers and standard financial accounts. An asset as collateral in escrow is not really the same as an individually owned asset, as another party has a contingent claim on it. A special accounting is required. Likewise, it is natural to think of the flow of future wage payments as human capital on the balance sheet, which is a standard view in some sense (Aiyagari 1994; Huggett and Kaplan 2015). Smart contracts on the distributed ledger allow an immutable commitment that future income flows be sequestered as collateral. A modified accounting needs to be implemented. These two components of distributed ledgers—accounts and contracts—are interacting.

6.3 Smart Contracts in Computer Science and Incentives in Economics: Contrasts, Similarities, and Problems in Implementation

Though the language of smart contracts and the language of mechanism design fit remarkably well with each other, there are some key differences. Many of these differences came up in the previous section but bear repeating. In economics we try to avoid *a priori* classifications of trust. Individual agents are self-interested and optimize given their information and incentives, whereas nodes in computer science are trustworthy or not, or trusted up to a limit. Their role is often stated in simplistic terms—to validate whether or not a transaction is valid. Insofar as possible, we should be clear about this difference across the disciplines and take advantage of the strength of the two approaches. This

means taking into account economic incentives in validation and potentially separating functionality into segregated components. There remain some key challenges in the interface.

To begin, there are other related differences: In economics it is a natural part of the design to give incentives to agents to be honest and obedient, as a function of the social context—that is, the purpose for which one is using the ledgers. In economics, writing out contracts as fully as possible is largely regarded as desirable, if it is possible.[6] In computer science, though, code is acknowledged to be "buggy," and hence it can be argued that simplicity has a special virtue. In economics, commitment is desirable if attainable, and renegotiation and bargaining are likely to change outcomes in an adverse way. In computer science, with its tendency to seek consensus and validate transactions in real time, the door is open to limited commitment, hence to more limited arrangements. This problem can be alleviated by separating internal contract and external validation incentives.

As in Narayanan and Weinberg (2018), there is a sparsity of hybrid models that occupy this middle ground between Nakamoto's dichotomy—that is, honest as opposed to malicious nodes, on one extreme, versus all players being potentially strategic, as in economics, at the other extreme. Narayanan and Weinberg think of cryptocurrencies as mechanisms and propose a protocol that must incentivize compliance for miners. They note that, for the most part, existing protocols are not incentive compatible. We provided some examples of this literature on the incentives for miners earlier.

6.3.1 Coding and Implementation Costs

On the downside, smart-contract coding can contain errors and has been difficult to use. Remedies are under development: Agrello is an initiative for human-readable smart contracts; iOlite has been developed to write smart contracts in natural language.

Smart contracts can be costly to implement, especially if initial validation of code, messages, and datasets are all put out for validation. In particular, calculations of gas charges to cover electricity costs in proof of work for Ethereum appear daunting.

Here is an example, from Nicolas Zhang (2019), focused on the coding cost. For context one can refer back to figure 2.1. There are two agents, *A* and *B*, with off-chain balance sheets that are not known to the other party. They face some balance-sheet shocks throughout time. If one agent is facing a positive shock, then there are excess reserves to invest. If one agent is facing a negative shock, then some liquidity is needed. The two agents wish to write a global smart contract that will function over time as situations evolve. They wish to implement the solution (a transfer function) on-chain.

The two agents first design a smart contract. Furthermore, imagine the two parties share their utility functions truthfully. This leaves an optimization problem to be solved, and the solver is the smart contract. Here, this seems a bit artificial. If the goal is borrowing/lending on the part of a first party with a second party only, then the desired amounts for borrowing or investing that the first party wants would be sufficient, as the other party always goes along (assuming no default). But in more complicated situations (e.g., a two-period, hybrid borrowing/lending insurance contract), the two agents would need to calculate what they want to do. That is carried into the example here, though, again, in a simplified context.

Suppose the solution to the borrowing/lending problem is not analytic (i.e., is not a closed-form solution and is solved by first-order-condition approximations). The derivatives are approximated by small, discrete differences depending on the size of a grid. The number of subtractions to form these differences is the dimensional of the discretized function domain (less one), and we are ignoring division.

In Ethereum there are costs to computing the derivative this way

(a) To form differences: gas × the dimensionality of the discretized function domain grid.

Next, try to set the derivative close to zero, hence

(b) To find local optima, a set of equality comparisons: 3 gas × the dimensionality of the discretized function domain grid.

Next

(c) To find the global optimum, a set of greater than equal comparisons: 2 gas × number of local optima.

Plus, in Ethereum, there is a one-time cost of implementing the smart contract. Add some cost during the life of the contract when it is called (for instance, updating every week, of every quarter, depending on the frequency the agents want).

In summary,

G_{create}3200 paid for a CREATE operation.
G_{create}200 paid per byte for a CREATE operation to succeed in placing code into state.
G_{call}700 paid for a CALL operation.

Given that the gas limit on Ethereum is 8,000,000 gas, even for something as easy as a borrowing and lending contract with a 1000-point discretized grid for the utility function domain, only 30 to 50 specifications of utility would fill an Ethereum block.

Thus, for more complicated optimization problems running only a few contracts (number of realized economic environments), blocks and costs would be higher. Of course, these costs come down significantly under alternative smart-contract protocols.

The next step is the sending of messages under the contract. Again figure 2.1 can be helpful. These messages can be verified

on-chain as in the validation protocols, now an exact application. There is some cost, especially with proof of work. However, the messages themselves would be incentive compatible, by design of the optimization problem with appropriate constraints, so the validation has simply to do with whether messages are faulty or not received. We turn to this in the material below.

Finally, some transactions data (e.g., data of past messages) can be secured off-chain. After initial creation, the document is encrypted and validated on a blockchain so there is a time-stamped immutable record to prevent tampering. Any attempt to change the underlying database relative to the original would be apparent even if the change itself were not evident. One particular instance is the InterPlanetary File System (IPFS), a peer-to-peer distributed ledger system (IPFS 2019). A file, and all of the blocks within it, is given a unique fingerprint, which is its *cryptographic hash*. To look up a file to view or download, the network finds the nodes that are storing the content behind that file's hash. Each network node stores only content it is interested in, plus some indexing information that helps figure out which node is storing what.[7]

6.3.2 Reliability of Messages

Another potential problem has to do with the reliability of messages. A relevant insight: The revelation principle can be established in many contexts even though message transmission and receipt can both be noisy; despite the noise, one may as well let the computer transmit the message to the network the way it would have been sent originally, under the original game and message space.

Prescott (2003) extends the revelation principle to the situation of zero communication in a principal-agent context (i.e., no message is sent at all). This impacts the design of the incentive contract and is in some sense a worst-case scenario, but it does completely avoid the cost of validation.

Going in the other direction, the implementation problem of Harris and Townsend (1981) is a way of letting common private information be public by having multiple agents announce underlying states. Without noise in messages, at least, having two agents with common information announce that information is sufficient, in a well-designed payoff matrix, to make information public. Intuition would suggest that if messages are noisy, more agents announcing is better, but of increasingly limited value.

This brings us back to the notion of fault tolerance, which allows for a fraction of nodes to communicate inaccurately, as in distributed computing, either through machine error or, in the computer science literature what often seems to be taken as equivalent, because the node is deliberately malicious. Here we will discuss in more detail this distinction, which can be key.

The basic problem is a version of the Byzantine Generals Problem. Two generals must coordinate an attack for it to be successful. One sends a message to the other but cannot be sure the message has arrived. So the second general, knowing this, sends a confirmation, but in turn cannot be sure it has arrived. No finite number of iterations is ever sufficient for there to be complete certainty. In another version, the message may be received but inaccurately. In a third version, the message is accurate but the message could have been sent by a traitor, again mixing up computer inaccuracy with malicious behavior.

Rubinstein (1989) focuses on and formalizes a closely related electronic mail problem as a coordination problem. He draws a connection to the Byzantine Generals Problem in a concluding section. Two players have to play one of two possible coordination games. Only one player receives information about the coordination game to be played. If the players follow an obvious intuitive-communication protocol, then it is impossible for there ever to be common knowledge about the true state. Furthermore, the situation with "almost common knowledge" after

a large number of successful messages remains different when compared with the coordination game played under common knowledge. This counterintuitive result shows that formalization with the logic of game theory can lead to surprises.

Morris and Shin (1997) build on this idea, drawing a key distinction between strategic and nonstrategic behaviors. Here is their version of the Byzantine Generals Problem, which is close to Halpern and Moses (1990):

> Two divisions of an army, each commanded by a general, are camped on two hilltops overlooking a valley. In the valley awaits the enemy. The commanding general of the first division has received a highly accurate intelligence report informing him of the state of readiness of the enemy. It is clear that if the enemy is unprepared and both divisions attack the enemy simultaneously at dawn, they will win the battle, while if either the enemy is prepared or only one division attacks, the attack will be defeated. If the first division general is informed that the enemy is unprepared, he will want to coordinate a simultaneous attack. But the generals can communicate only by means of messengers and, unfortunately, it is possible that a messenger will get lost or, worse yet, be captured by the enemy. (Morris and Shin 1997, 174)

Morris and Shin (1997) proceed in steps. They begin, as did Rubinstein (1989), with an imposed, simple, naïve communication protocol and a fixed objective. The fixed objective, the occasionally successful coordinated attack, though this may not be achieved, is defined as follows: (1) it is never the case that an attack occurs when the enemy is prepared; (2) it is never the case that one division attacks alone; and (3) both divisions sometimes successfully coordinate an attack.

The protocol allows back-and-forth communication: If the first division general hears that the enemy is unprepared, he sends a message to the second division general with the instruction "attack." If that first message arrives, the second division general sends a messenger with a confirmation that the first message was safely received. If the confirmation is delivered

without mishap, the first division general sends another message to the second division general informing him of this fact.

To fill out the structure, suppose that with probability $\delta > 0$ the enemy is prepared, while with probability $1 - \delta$, the enemy is unprepared. The first division general knows which of these two contingencies is true, while the second division general does not. Each messenger gets lost with independent probability $\varepsilon < \delta$. Suppose that a successful attack has a payoff of 1 for the generals, while an attack that is unsuccessful is disastrous (either because the enemy is prepared or only one division attacks) and has a payoff of $-M$ for the generals, where M is very large. An action protocol for the generals is optimal, given the communication protocol, if it maximizes the generals' expected payoff.

The first result: If the communication system is sufficiently reliable, then the optimized action protocol coordinates an attack almost always when the enemy is unprepared. Specifically, the first general attacks when the enemy is unprepared, even if he has not received a message confirmation from the second general. The second general attacks if he has received one message from the first general. Morris and Shin (1997) show this sequence dominates any other action protocols. For example, it dominates when the first general always attacks when the enemy is unprepared and the second always attacks, and also dominates when each general attacks if the second receives a message from the first and the first has received a confirmation of that message from the second.

But the above optimal action protocol turns out to be sensitive to a natural strategic concern, formalized as an equilibrium outcome in an incomplete-information game, with payoffs specified for each party as a function of that party's action and the action of the other general. In particular, off-diagonal elements give a payoff of $-M$ to the general that attacks alone regardless of whether the enemy is prepared. The previous

protocol can be relied on if the generals choose to follow it. Is this what trusted nodes do? Nodes that are not (even) malicious? Not necessarily. Again from Morris and Shin (1997), suppose that the first division general knows that the enemy is unprepared, sends a message to the second division general, but receives no confirmation. He may be tempted to not commit his division to the battle in these circumstances as the probability is greater than one-half that his message was never received. In turn, given this and following the natural logic, the second division general may hesitate to attack if he has not received a reconfirmation from the first division general. In the strategic coordinated-attack problem with the naïve communication protocol, neither general ever attacks. Ironically, this happens if the communication system is sufficiently reliable.

What is the intuition? Each general must have an incentive to attack once he is actually called on to do so. Generals in the strategic game are not allowed to *commit* to strategies before the communication stage. Second, the generals have *different* objectives—the first division general would much rather that the second division attack alone than that the first division attack alone. Both these features are necessary for the paradox.

Relatedly, the communication protocols matter. Morris and Shin (1997) consider a revised "simple communication protocol" that builds in a kind of commitment. Suppose that if the enemy is unprepared, the first division general sends one message to the second division general informing him of this state of affairs. The second division general sends no confirmation (i.e., he is not allowed to communicate back). For sufficiently small ε, a coordinated attack almost always occurs whenever the enemy is unprepared.

This then raises the obvious larger issue: If one could design the communication system, subject to unavoidable communication errors that would be used by players in a strategic environment, how would it be done? Chwe's (1995) notion of

"strategic reliability" is consistent with the discussion above: The design goal shifts from maximizing connectedness (or other measures of network reliability) to maximizing the likelihood that agents communicating over a flawed network can achieve desired outcomes via equilibria in the induced game. Multiple channels can facilitate collective action via redundancy, the sending of the same message along multiple paths or else repeatedly along the same path (Chwe 1995; De Jaegher and van Rooij 2011).

Coles and Shorrer (2012) examine how this logic extends to multiplayer settings where one informed agent serves as a "leader," relaying messages to and from the other parties. They show that the multiple channels may permit collective action: Parties may be able to coordinate their actions when messages arrive at their destinations in a sufficiently correlated fashion, as they would be if transmitted from a common sender. This also permits cutoff equilibria, where players take action after receiving a minimum number of confirmations.

In summary, though the consensus protocols outlined in chapter 5 deal with trust and seem to allow nontrusted or malicious parties, motivated by the Byzantine Generals Problem, they are not necessarily robust to natural strategic considerations. That is, they do not take the step of formalizing validation as strategic behavior in an incomplete-information game and may not be implementing an optimally designed communication system.