Innovative financial designs utilizing homomorphic encryption and multiparty computation

Robert M. Townsend (rtownsen@mit.edu)
Elizabeth & James Killian Professor of Economics, MIT

Nicolas XY. Zhang (nxyzhang@mit.edu)
Computer Science and AI Laboratory, MIT

August 23, 2022

Abstract. Encryption - the process of encoding information - mitigates the damage that comes from obstacles to exchange: private information, limited communication, and limited commitment. We emphasize specifically that encryption is a way to implement optimized solutions to bilateral and multi-agent mechanism design problems as smart contracts. To illustrate how such encryption schemes could be relevant to economic applications, we first illustrate how "classical" problems such as auction design with private values, constrained-optimal hybrid insurance and credit schemes for markets suffering from liquidity problem. The common thread in each of these applications is that messages are kept secret from other agents in a contract and from third parties, communication channels are optimized, and data are secured and immutable. Agents can commit to arrangements and commit to the way they are implemented, without inconsistencies, even in evolving situations where they would have liked to renege. We make use in particular of homomorphic encryption, HE, and multi-party computation, MPC, which we will describe in detail in the text and concretely implement for the auction with private values and the constrained-optimal hybrid insurance and credit schemes for markets suffering from liquidity problem.

(1) Our design methodology: understanding cryptography technologies, and leveraging them (selectively) as new design possibilities

Financial systems can be viewed as systems allowing agents to enter into contracts. However, contracts are not-single dimensional (e.g., more or less money); the more contingencies financial systems can deal with, the more efficient and resilient they become. These contingencies are notably constraints generated by (1) untrusted messages - crucial for instance in the counting of ballots, or of auction bids -, by (2) unobserved actions - exacerbating risks from counterparties, settlements, and commitments), or by (3) unobserved states - while estimating competitors' bidding power and valuation of the good sold in an auction for instance. We will see in a range of applications how these issues can be raised, and how uncertainties, commitment issues and secrecy cause distortions.

Encryption, as the science of protecting messages' content and authenticity, while securing senders and receivers' identities and privacy, enables a large toolbox for economic design. We will then see how additional guarantees on authenticity, on commitments and enforcement of contracts can help solving the issues raised above. Moreover, we will also study how a wider range of public-privacy states of information and the ability to perform computations over encrypted data (thus preserving their secrecy) can fundamentally improve the design of financial contracts in linking agents' differing preference and risk profiles (the unobserved states that play a crucial role in mechanism design). We will thus be able to introduce flexible hybrids contracts, in between pure borrowing/lending and pure insurance contracts.

We follow three steps in the design of these new economic systems. First, we decompose and examine separately the different technological components of encryption, to "translate them" into individual tools formulated within the usual mechanism design environment (described in the following section). Second, having these new economic design tools at our disposal, we "revisit" several classical economic problems (such as the ones mentioned above - auctions, borrowing/lending, insurance...) and craft the incentives and constraints that have to be put in place or lifted for an improved contract to be designed, in an objectives-first and context dependant approach (there can be different goals and values for a contract to aim to implement, and each economic application presents a different environment for the contract to take into account). Finally, we show how this contract devised in the abstract language of mechanism design can be implemented in real life using different combinations of existing technologies - each of them implementing the correct set of incentives, but potentially presenting different engineering trade-offs that we try to highlight. This way, our contribution will be to present an end to end "cookbook" on cryptography technologies to implement several mechanism design solutions. We hope to provide inspiration and methods for economic and policy designers to be able to leverage the right cryptography tools to design the right solution to the specific problems they will be facing.

(2) The Mechanism Design Framework

In general, at an abstract level, an economic application is laid out in terms of preferences of the agents, their endowments, and the technology available to them. In fact, to examine the extent to which different historic forms of economic contracts and organizations can be explained by information-incentive problems, and to explore how to improve these contracts and organizations, it is useful first to retreat to the abstract setting of the exchange and endowment economy, in which we add constraints related to the flow of information - for instance that an agent's preference is fully private and not observable, or that his endowment or output are not observable (at least not without considerable cost). These information related constraints are key in *demonstrating* how a seemingly imperfect form of organization was actually the best implementable possible, given the technologies of the time. Related, new technologies that allow relaxation of some of these constraints on the flow of information (such as homomorphic encryption and multiparty computation explored in the next subsection, which allow computations to run in a privacy-preserving fashion, guaranteeing each individual input's secrecy) can potentially lead to better implementable mechanism design solution. Townsend JME 1988's findings on limitations introduced by publicly communicated messages provides for instance guidance on the gains that could be collected from using privacy-preserving computation technologies.

(3) "Decentralizing" third parties through homomorphic encryption and multiparty computation

This solves problems arising from the unobserved states mentioned above. We will lay out some mathematical principles key to MPC and HE here, and reference these in the detailed examples in subsequent sections to highlight how it translates to engineering design.

The key property of homomorphic encryption, HE, is that a function f can operate on a true underlying space or equivalently operate on the space of encrypted (i.e. encoded) values. That is, in order to utilize a function f in some application, one does not need to input the "plaintext" information which reveals the original and true values of a given agent, but rather, one can encrypt the agent's data, so that data can be kept private, apply the same function f on encrypted input data, get a result in the encrypted space, and all of this is the same as if the message had been true private values and the output from f were encrypted. That is f(Enc(m)) = Enc(f(m)) . Equivalently, what we want in normal space is on the right but the way to get it is in the encrypted space on the left of this equation: Decipher[f(Enc(m))] = f(m). Note that this is for one agent, one message at a time, in contrast to MPC below for multiple agents (but which is less flexible). Further, **the "distributivity" of the encryption operation with the function operation gives flexibility in navigating between the normal space and the encrypted space** (see concrete examples of such "navigation" in sections 3, 4 and 5). The distributive property is telling us what rules manage the parenthesis that order the sequence of operations - a distributive scheme allows operations outside the parenthesis to be "distributed" to the elements within, whereas a non distributive scheme doesn't allow such treatment of the parenthesis. This allows us to perform a series of composed transitive functions g all on top of the encrypted message, as on the left: g[f(Enc(m))] = g[Enc(f(m))]. Thus, the agent or contract performing all the functions never gets to see the actual content of the message, nor the true outcome of f in the normal space (in subsections sections we will write in parenthesis *** for "distributivity"** next to equations in which this property is used).

The key property of multiparty computation, MPC, is that rather than determine the value of a linear function f using inputs values from multiple agents, seeing the inputs and the result of f directly, in normal space, one can run the function f on encrypted private inputs, then decrypt the output value or relevant summaries, and finally share that result. Namely, for J agents Decipher$[f(c_1, c_2, ..., c_J)] = f(m_1, m_2, ..., m_J)$ where $(c_1, c_2, ..., c_J) = Enc(m_1, m_2, ..., m_J)$ are the encrypted values of the inputs from m different agents. Distributivity of MPC fails, and we can't compose transitive functions g on top of f - ie Decipher$[g(f(c_1, c_2, ..., c_J))] \neq g(f(m_1, m_2, ..., m_J))$. But combining MPC with HE, we have what MPC means plus the distributivity in operations - ie the inequality above becomes an equality.

Further, another important way that MPC techniques differ from HE+MPC is in the distribution of the work to perform the computation among the participating parties. In classical MPC constructions (Goldreich et al. 1991), the work of the MPC protocol is evenly distributed between all participating parties. The symmetry in the MPC protocol might make installing, maintaining and running the MPC software on each agent's internal machine more challenging - as cheaper machines from one agent would slow down the overall computation for all. With HE in addition of MPC, the distribution of work in the protocol can be very uneven, with in extreme cases such as in de Castro et al 2020 all of the computation work happening on one single machine (each agent still performs the encryption steps on his own machine). In addition, the work done on the agents' machines is largely independent of the actual computation done on the encrypted data, which allows more flexible updates while maintaining of the computation functions.

Lastly, an important difference between classical MPC and HE-MPC concerns decryption - the symmetry of computation workload between all J agents in classical MPC requires all J agents to agree for the result to be decrypted. HE gives on the other hand flexibility in choosing how many agents need to agree in order for the result to be decrypted. Any number of agents k between 1 and J can be chosen - prior to the computation - to allow for decryption. For illustration, here below would be a list of the main steps followed by each agent in a HE-MPC computation, such as that of de Castro et al 2020. **Overall, let's note that with HE and MPC, encrypted inputs can be kept private and still contribute to operations performed on top of them. We will call that a "private-but-contributing-state-of-information", which we will make full use of below.**

1. Each agent individually generates its own key pair, where each key pair contains a public encryption key and a private decryption key (**related to our public-key cryptography section above**).
2. All agents submit their public keys to each other (**classic MPC**) or to a central server or an agent chosen to perform the homomorphic operations (**HE-MPC version**). That entity will be called the "pseudo agent" described in subsequent sections.
3. The pseudo-agent (**HE-MPC version**) or all agents through interactions (**classical MPC version**) combines all agents' public keys into a single joint/shared public key (**the core technique on which all forms of MPC are built**).
4. This new joint/shared public key is distributed to all agents (**the core technique on which all forms of MPC are built**).
5. Each agent encrypts its private data using this new joint/shared public key, generating a ciphertext (an encrypted block of data) that cannot be decrypted by anyone but themselves (**related to the public-key cryptography section above**).
6. Each agent sends the ciphertext of its private data to the pseudo-agent (**HE-MPC version**) or to each other (**classical MPC version**). This ciphertext completely hides the agent's data.
7. The pseudo-agent (**HE-MPC version**) or all agents together through interactions (**classical MPC version**) run computations on all the encrypted data, producing an encrypted result of the computation (**that is the homomorphic encryption part. In classical MPC computations is distributed "in a clever way" among all agents, instead of homomorphically - see for instance Goldreich et al 1991**).
8. The encrypted result are sent back to each agent.
9. Each agent uses the private key they generated in Step 1 to partially decrypt the answer (which is still scrambled at that point) (**related to public-key cryptography**).
10. Each agent sends this partially decrypted answer back to the pseudo-agent (**HE-MPC version**) or to each other (**classical MPC version**).
11. The pseudo-agent (**HE-MPC version**) or all agents together through interactions (**classical MPC version**) combines the results of all the partial decryptions it receives from agents to produce the decrypted result that is then shared back with the agents (**in HE-MPC the server just needs agreement from k between 1 and J the total number of agents to decrypt. In classical MPC all agents need to combine all partial decryptions to produce the decrypted result - which all J agents would see, which is not the case in HE-MPC**).

Figure 2: main steps followed in an HE-MPC computation

Over all, what private information to reveal (or not) even as a computation is run on it, and who can decide of it, become a new choice element of mechanism design. In subsequent examples - which are one specific instance of HE-MPC, and tailored to the mechanism design solution so that it might presents variations from the steps listed above - we take pains to clarify exactly what is needed, utilizing the best available technology. We do not force financial applications onto a given technology, but go the other way around, starting with the economics.

(4) An example of improvement replacing the trusted third parties - auctions without auctioneers

To see an online interactive tutorial guiding through this example, please visit the following Jupyter Notebook http://bit.ly/FHEauctions

Generally, auctions are used frequently but are organized typically by a third party, as in Mortgage Capital Trading's use of a Trade Auction Manager. However, trusted third parties are not always needed, and not always beneficial. In the previous example of current bid-wanted-in-competition schemes, BWIC, dealers are being asked to bid on listed securities. A typical potential abuse: run over the telephone, the seller of the asset in question who is the organizer of the auction tells buyer A : "I prefer you my friend, but buyer B just offered a slightly higher price, so if you can just make some effort the good is yours". Vice versa with buyer B to prop up the price. Neither buyer A nor B can check the bid of the other buyer. Some of China's P2P platforms also presented misleading information to investors or engaged in AI cream-skimming in the creation of securitized pools. Trumid, on the other hand, markets itself as a Wall Street company with an algorithm intended to replace OTC dealers who are thought widely to trade on the proprietary information of their clients.

However, companies like Trumid are "replacing" OTC dealers by being more efficient and better trusted third parties. However, we argue here that these improvements could be brought further, by totally "getting rid" of the trusted third party. There are for instance several ways to implement our auction-without-auctioneer scheme with HE and MPC. Let's give one example here, building on lattice cryptography (so that this is post-quantum secure, ie secure against attacks from quantum computers, when they will exist). For the purpose of our exposition here to be self contained, let's simplify some of the cryptography elements - notably those related to the Ring-Learning with Errors (RLWE) model. Let us just note that many of the mathematical properties we make use of, such as commutativity and distributivity, comes from those of mathematical Rings. For simplification purpose, let's see all elements drawn from these Rings as polynoms, for which the commutativity and distributivity properties are verified. Indeed, if one takes the key pair generation and decryption steps as given (steps 1 to 4, and 8 to 11 in figure 2 above) all other steps are just like "classical" public key cryptography, with our examples focusing on the math/mechanism design of the computations ran by the server in step 7. For the avid reader more details can be found below, in the optional subsection, which we adapted from Brakerski (2012) and Fan and Vercauteren (2012) (hereafter and in the literature called the BFV scheme). The specific key generation protocol and corresponding decryption protocol we make use follows the work of Asharov et al. (2012), of which no understanding is necessary here to follow the mathematical operations we describe below.

(Optional read) Distributed Key Generation (steps 1 to 4 in figure 2)

Let's run our example with two agents, A and B, who want to see whose bid is higher and whose is lower, without relying on a trusted third party but without revealing to each other the exact value of each one's bid.

Consider the following standard BFV key generation protocol, which outputs a public-key (pk)/secret-key (sk) pair of the form

$$(pk, sk) = \big((a, \delta), (\ s, e)\big) \tag{1}$$

where a and δ are two uniformly random samples over the ring $R_q = Z_q[x]/f(x)$ where $f(x)$ is some degree-n polynomial. The error term e is sampled from a discrete Gaussian with large standard deviation (such noise terms are sometimes referred to as 'flooding' or 'smudging' terms). δ is a public scaling factor to prevent the small error term e from corrupting the message m (Ring-Learning with Errors, RLWE). Our key generation protocol relies on the common random string (CRS) model and begins with the assumption that all participants have access to the same truly random seed for a pseudorandom number generator (PRNG). All participants then use this seed to generate the same pseudorandom sample a from R_q. Each party i then generates the following key pair:

$$(pk_i, sk_i) = \big((a, \delta), (\ s_i, e_i\big) \tag{2}$$

So agent A has the following key pair :

$$(pk_a, sk_a) = \big((a, \delta), (\ s_a, e_a)\big) \tag{3}$$

And agent B the following key pair :

$$(pk_b, sk_b) = \big((a, \delta), (\ s_b, e_b)\big) \tag{4}$$

Once all these public keys are generated, the shared public key is computed by summing over $\mathbb{R}_q$:

$$pk = \big(a,\ a \cdot \sum_i s_i + \sum_i e_i\big)$$

The result of this sum is a well-formed public key for the BFV scheme, where the corresponding secret key $= \sum_i s_i$ is already distributed in additive shares among the participants.

Multiparty encryption of each agent's bid (steps 5 to 11 in figure 2 above)

Now that each agent has its unique secret key, while sharing a public key, they can start using these to encrypt their bids. To describe this encryption process we will follow a sequential description, even though the actual order of the sequences don't matter (both can even happen at the same time).

So A (let's say A starts, even though it would be the same if it's B first) : A sends to B its bid m_a, obfuscated by its secret key s_a, in the following encrypted message (let's call it ct_a , for ciphertext from A) :

$$ct_a = a \cdot \ s_a + \delta \cdot m_a + e_a \tag{5}$$

B cannot guess (from the security proof of the BFV scheme) the value of A's bid m_a. He sends similarly his encrypted bid m_b to A (let's call it ct_b , for ciphertext from B) :

$$ct_b = a \cdot \ s_b + \delta \cdot m_b + e_b \tag{6}$$

Step 2. Multiparty addition of each agent's bid

Let's make each agent "add in" his own secret key to the encrypted message he received from the other party. Ie, for agent A, do the operation

$$a \cdot \ s_a + e_a + ct_b \tag{7}$$

and for agent B the operation

$$a \cdot \ s_b + e_b + ct_a \tag{8}$$

So now agent A has the new ciphertext, by writing

$$e_a + e_b = e, \tag{9}$$

and:

$$s_a + s_b = s \tag{10}$$

$$ct_{b\prime} = a \cdot \ s + \delta \cdot m_b + e \tag{11}$$

And agent B has the new ciphertext

$$ct_{a\prime} = a \cdot \ s + \delta \cdot m_a + e \tag{12}$$

And, if both of them resend the ciphertext each one had of the other, then both can do the operation

$$ct_{a\prime} - ct_{b\prime} = ctFinal = \delta \cdot (m_a - m_b) \tag{13}$$

The sign of which reveals which agent's bid was higher at the start.

Let's note that here with only 2 agents this is a limit case that would reveal, by difference, each agent's bid value to each other. For strictly more than 2 agents, this wouldn't be a problem unless all but one agents coordinate and share their inputs, in which case they can also infer the input from that last agent as well. For just 2 agents, let's see now how δ can be a value kept secret to each other, hence hiding the real value of both agent's bid.

Using the 2 agents case to introduce the concept of "pseudo agent" nodes

We have the previous setting with agents A and B. Let's now introduce a "pseudo agent" C, which can be for instance a server that has been jointly set up by A and B, and then left as a "deterministic box" on its own (we use the "deterministic box" image of some box that has been pre-coded to do some things, and that then can be guaranteed to do it, even if no one audits what is going on inside). In a distributed ledger language, C would be a smart contract node. But here we present a more general case than just DLT-based smart contract. This node will be very useful in the generalization work presented later in the document.

A and B still have the following key pairs :

$$(pk_a, sk_a) = \big((a, \delta), (\ s_a, e_a)\big) \tag{14}$$

and

$$(pk_b, sk_b) = \big((a, \delta), (\ s_b, e_b)\big) \tag{15}$$

However, now in **step 1** both A and B send their ciphertexts to C, instead of to each other. C also asks both A and B to send the following combination of A and B's public and private keys (let's call them nn_a and nn_b for "not so random noise from A" and "not so random noise from B") :

$$nn_a = a \cdot\ s_a + e_a \tag{16}$$

and

$$nn_b = a \cdot\ s_b + e_b \tag{17}$$

nn_a and nn_b have sufficient cryptographic properties (as in BFV) to conceal A and B's private keys (s_a, e_a) and (s_b, e_b), while allowing C to decrypt ct_c , where ct_c is obtained through C adding the two ciphertexts ct_a and ct_b he received from A and B. I.e.

$$ct_c = ct_a + ct_b = a \cdot\ s + \delta \cdot (m_a + m_b) + e \tag{18}$$

Indeed, we have :

$$nn_a + nn_b = a \cdot\ s + e \tag{19}$$

so that

$$ct_c - (nn_a + nn_b) = \delta \cdot (m_a + m_b) \tag{20}$$

Similarly, C can just do the operation

$$ct_a - nn_a - (ct_b - nn_b) = \delta \cdot (m_a - m_b) \tag{21}$$

And, since we assume that C has "pre-coded" in him :

- "STEP1 TAKE INPUT 1 FROM A, SUBTRACT INPUT 2 FROM A. (ie equations 18 - 16)
- STEP2 TAKE INPUT 1 FROM B, SUBTRACT INPUT 2 FROM B. (ie equations 18 - 17)
- STEP3 TAKE RESULT FROM STEP1, SUBTRACT RESULT FROM STEP2. (equation 18x2 - (16+17) = equation 21).
- IF RESULT FROM STEP 3 IS POSITIVE THEN AGENT A WON THE AUCTION. ELSE AGENT B WON THE AUCTION."

Then C can just announce to both A and B who won, with A and B never seeing anything else than the "not so random noises" and ciphertexts each of them sent to C, which do not reveal their bid values.

Going back to the BWIC auction example, in which the seller of an asset organizes an auction scheme, one could apply the homorphic encrytion scheme we laid out above to limit the seller's incentives to misreport. Then, each buyer's bid will be encrypted by him using his unique secret key, to prevent anyone else from tampering with it (especially the seller/organizer of the auction, who agrees to not possess each buyer's individual secret key). Furthermore, the organizer of this BWIC auction agrees to only provide the matching (or ranking) operation of this auction, performed homomorphically on top of each buyer's encrypted bid, as "pseudo-agent" C above does. The encrypted bids thus protect buyers' privacy in both keeping it untampered, and by keeping them anonymous, to reduce potential conflict of interest and collusion between the organizer of the auction and some participants (if the public/private keys are allocated randomly to each participating agent).

Some decryption may come at the end if the winner needs to be revealed. The asset purchased might be on a distributed ledger with the transfer executed under a smart contract, but this part is separate and is not necessary per se. An alternative, a third party might be trusted to make the asset transfer. Both are an option, a choice for possible auction designs.

(5) An example of improvement replacing the central planner - in flexible, hybrid contracts between borrowing/lending and insurance (Townsend, JME 1988)

In the previously described auction scheme, the secrets, the underlying messages before encryption, need never be revealed to other agent or an institution performing the matching/ranking operation. We now extend this "private-but-contributing-state-of-information" to tackle one of the core problems in mechanism design - that surrounding the treatment of private information held by agents, which are needed to convince its counterparties of its truthfulness and secure their commitment, but which cannot be fully revealed for fear of some counterparties exploiting the information.

One such example is that of the borrowing/lending and insurance literature, as described in Townsend JME 1988's findings on limitations introduced by publicly communicated messages. Townsend JME 1988 also provides guiding on the gains that could be collected from using privacy-preserving computation technologies, and introduces flexible contract between bor-rowing/lending and insurance, hybrid in that it has aspects of both. In interbank lending markets for instance, banks try to buy or sell specific amounts of liquidity, that are linked to their funding needs. If all banks real funding needs were known, some optimal distribution and pricing schemes could be devised. However, if anyone of these banks accessed some others' real funding need, they would gain negotiation power and could exercise leverage with this knowledge, or sell the information to others. The context also includes a variety of payments arrangements. For example, money transfer organizations run short of liquidity, of one object or another, and would benefit from a more organized or improved market. In Kenya, agents for Safaricom are frequently running out of fiat Shillings or M-Pesa, seeking gifts or loans informally from other agents. In Indonesia agents acting on behalf of banks in towns and rural areas are also typically short of liquidity, want to rebalance, but complain it is difficult to do so. Broker dealers as agents in New York markets have their own liquidity problems, and despite borrowing and lending though repo markets, participation is segmented and interest rates have moved erratically, counter to policy rates.

Townsend, JME 1988 presents for instance a schematic case between two agents, one can be thought of as a potential borrower but with varying needs and the other who is on the other side of the trade. The lender who in turn has varying needs at the time of repayment. the two agents agree in a first, initial contract agreement stage, what the amounts that will be borrowed in the second time step, depending on the realization of a shock. However, to keep privacy as to whether the borrower is in a high or low liquidity need, "a layer" of scrambling will be added for the lender not to be able to infer for sure if the borrower is in the high or the low liquidity situation. Each agent is seeing the outcome, the allocation of funds but not the messages that triggered that outcome, other than their own. Please see below a table summarizing what the environment, the "scrambling", and the contract is (from Townsend 1988). This scheme exhibits the full power of encryption, as here it is even harder to conceal agents' "bids" than in the auction case presented before.

Partitioned private ledgers and the gains from concealment.

Private information solution					
θ_0^a	(c_0^a, c_0^b)	$\pi(c_0, \theta_0^a)$	θ_0^a, θ_1^b	(c_1^a, c_1^b)	$\pi(c_1, \theta_1^b)$
0.2	(1.75, 8.25)	1.0	(0.2, 0.2)	(4.75, 5.25)	1.0
			(0.2, 0.9)	(2.0, 8.0)	1.0
0.9	(0.0, 10.0)	0.1159346	(0.9, 0.2)	(3.75, 6.25)	1.0
	(1.75, 8.25)	0.0339681			
	(3.25, 6.75)	0.8544384			
			(0.9, 0.9)	(1.0, 9.0)	0.86106
				(10.0, 0.0)	0.13839

Agent a has preference shocks, urgency to consume, θ_0^a, at date $t=0$ of either 0.2 or 0.9. The announcement triggers consumption allocations (c_0^a, c_0^b) for agents a and b, the latter as the second party. These are listed in the second column of the table. Notice that if the announcement is 0.9, then a lottery puts about 3.3% probability on (1.75, 8.25) the deterministic allocation when 0.2 is announced. Probabilities $\pi(c_0, \theta_0^a)$ are in the third column of the table. At $t=1$, agent b announces 0.2 or 0.9. The mechanism knows the previous history of incentive-compatible announcement of θ_0^a at $t=0$, but agent b does not. The agent has an incentive to announce θ_1^b truthfully, and there is an insurance transfer along all paths. If, however, agent b were to have seen θ_0^a, there is no insurance over θ_1^b (top row right). Here, without that information, agent b might be tempted to claim urgency, 0.9 always, but doing so risks losing everything with 14% probability if in fact agent a had announced 0.9 (as in the bottom row, last column). In summary, when $(c_0^a, c_0^b) = (1.75, 8.25)$ is observed, agent b remains uncertain of the type agent a reported in $t=0$, and this is crucial for incentives.
Source: Townsend (1988).

One can view agent specific shortages in these contexts as idiosyncratic shocks, for which ex ante insurance would be good. But revealing those shocks over time too quickly can damage the possibility of beneficial trade. Though a simplified analogy, this is much as with standard insurance, in which ex ante insurance possibilities are lost once the ex post adverse event has occurred. This is known to be a problem in the foreign exchange markets in the sense that information on trades should not be revealed too quickly (Lyon, 1996).

These applications can be cast as mechanism design problems that feature concealment, that is keeping private messages private but utilizing these as inputs to generate outcomes. For example, two agents have utility functions subject to privately-observed preference shocks. The first agent in the first period, agent a, can be patient or urgent to consume, private knowledge, and likewise for the second agent in the second period, agent b. The 'planner', to use the standard language of mechanism design, sees the messages that each of the two agents sends, but makes sure the other agent does not. Incentives to tell the truth in the second period are easier to muster if the sender in the second period, agent b, does not know what message was sent by agent a in the first period. The information-constrained optimal allocation is such that lying in the second period generates a very bad outcome for the sender with positive probability.

But we do not rely on this mythical central planner or trusted third party. Let's examine now how one would build such a scheme without him.

Implementation notes for the hybrid borrowing lending case without central planner

Let's use the same setting as the previous auction between two agents : we thus have agents A and B, and "pseudo agent" C. We introduced "pseudo agent" C here to help conceal a new elements of privacy - we now want to protect the "actual message" sent by agent A at time step 2 (conceptually similar to protecting a bid value), but we also need to compare that value with preset values *which also have* to be protected (as we don't want B or C to be able to retrospectively guess from the value transferred what the "actual message" sent by agent A at time stage 2 - the first date of contract implementation - was. This is done through randomization as described above, but also through another layer of homomorphic encryption and decryption enabled by interactions between A and C, and between B and C, in addition of the interactions just between A and B. See for instance the summary schematic view we draw below detailing how messages exchange between A, B and C happen, and how information can be concealed by being encrypted by one agent but still used in computations by another agent even while being encrypted).

To summarize from above, but also elaborate, agents A and B negotiate a risk pooling contract carried out over two periods, time step 2 and time step 3. There is also an initial contract setting at time step 1, and at the end of which they put their initial assets as endowments into escrow. In period two A sends a message - either h_A, in the case A has a high need for liquidity, or l_A, in which case A has a low need for liquidity. The "central planner" as in Townsend, JME 1988 then takes this message, and requests money from B or vice versa accordingly (with some randomization going on if the message is h_A, for B not to be able to retrace with certainty what A sent in the first place). Please see sections 4 and 5 from Townsend, JME 1988 for a summary of theoretical optimal solutions and table 10 for empirical computations.

Here the "central planner" is replaced by the "pseudo agent" C. The scheme goes as follows :

Step 1. At time step 1

The contract to be executed has already been determined, here exactly the information constrained optimum in Townsend JME 1988. Agent A sends messages $Enc_A(h_A)$ and $Enc_A(l_A)$ to agent B, without telling him which ciphertexts correspond to which value (ie without telling him which order he sent the two messages - that is low first and then high or the other way around). This is in order to help conceal at time step 2 the value of the actual message m A will send - indeed, when B will receive m at time step 2 then he will have to differentiate m with h_A and l_A (please see the schematic view of the messages being sent in our protocol at the end of this section). By scrambling the order A can thus prevent B from knowing what the value of m is. By adding "pseudo agent C" in the protocol one can now split the information, as C would know the order in which A sent h_A and l_A to B, but not how to read $Enc_A[Enc_B(m_1$-m)] and $Enc_A[Enc_B(m_2$-m)] since C cannot decrypt A and B's encrypted messages. But C can perform homomorphic operations on it, such as using these values as encrypted *weights* in determining the encrypted final allocation value. This final allocation value will then be sent back to A and B for decryption, but without A and B being able to know what weights C used to come up with this value (see FIRST OPERATION, SECOND OPERATION and COMPUTE steps at the end of this section). B on his side knows the values of $Enc_A[Enc_B(m_1$-m)] and $Enc_A[Enc_B(m_2$-m)], but not what m_1 and m_2 correspond to.

Concerning the encryption, the subscript A after Enc means that it has been encrypted using A's key pair. So, recalling the notation of key pairs and encryption from last section :

$$Enc_A(h_A) = a \cdot\ s_a + \delta \cdot h_A + e_a \tag{22}$$

and

$$Enc_A(l_A) = a \cdot\ s_a + \delta \cdot l_A + e_a \tag{23}$$

Let's assume for simplification in this case that $\delta = 1$, so that the Enc operator we just define here is commutative, ie

$$Enc_B(nc_A(x)) = Enc_A(Enc_B(x)) = a \cdot\ s_a + a \cdot\ s_b + x + e_a + e_b \tag{24}$$

This is true for any message x. In practice schemes such as BFV can be made commutative, even if the mathematics behind become a bit "heavier". We will use this commutative property in the encoding at Step 2, Period 2, of the deterministic protocol "pseudo agent C" will follow.

From there on let's call $Enc_A(m_1)$ the first value agent B received, and $Enc_A(m_2)$ the second value agent B received. If A sent $Enc_A(h_A)$ first and $Enc_A(l_A)$ second then

$$Enc_A(m_1) = Enc_A(h_A) \tag{25}$$

and

$$Enc_A(m_2) = Enc_A(l_A) \tag{26}$$

and vice and versa. But B doesn't know.

Agent A also sends to B, if he sent to him $Enc_A(h_A)$ first and $Enc_A(l_A)$ second:

$$Enc_A((1,0)) = (a \cdot\ s_a + 1 + e_a, a \cdot\ s_a + 0 + e_a) \tag{27}$$

Else A sends to B, if he sent to B $Enc_A(l_A)$ first and $Enc_A(h_A)$ second:

$$Enc_A((0,1)) = (a \cdot\ s_a + 0 + e_a, a \cdot\ s_a + 1 + e_a) \tag{28}$$

As encrypted messages, these are uninterpretable by B. B then encrypts this message with his own key pairs, so that the message is encrypted using the "combined" multiparty key pair built using both A and B's pairs (similar to what we did for the auction without auctioneer example). B sends that to C. That message is then now

$$Enc_B(Enc_A((1,0))) = (a \cdot\ s_a + a \cdot\ s_b + 1 + e_a + e_b, a \cdot\ s_a + a \cdot\ s_b + 0 + e_a + e_b) \tag{29}$$

if A sent to B $Enc_A(h_A)$ first and $Enc_A(l_A)$ second, else it is $Enc_B(Enc_A((0,1)))$ in the reverse case. Let's call that message "C's input from B from STEP 1". Here the value 0 means null, and the value 1 means non-null as will be the case in the rest of the document.

Step 2. At Period 2

Agent A sends the encrypted version of its actual message Enc_A(m) (which can only take two values, either h_A or l_A) to B. Please note that this step is not mandated per se. Agent A can do whatever he wants. But the incentive compatibility of the mechanism ensures what A will do, as prescribed.

B then sends back to A for reference $Enc_B(0)$. This reference will be encrypted by A using A's own keys, and sent to C, to have an "encrypted null reference" encrypted by both B and A's keys, $Enc_A(Enc_B(0))$ (let's call it the "encrypted null reference" from now on). We will see right here already how this "encrypted null reference" will "show up" in the set of messages sent between B and A. Indeed B also sends to A the encrypted pair ($Enc_B(Enc_A(m_1)$ - Enc_A(m)), $Enc_B(Enc_A(m_2)$ - Enc_A(m)))= $Enc_B[Enc_A(m_1)$ - Enc_A(m)), $Enc_A(m_2)$ - Enc_A(m)]. Note that this is a vector (a pair). This pair's value is for instance Enc_B((0,1)) if $m_1 = h_A$ AND if m=h_A (again here the value 0 means null, and the value 1 means non-null as will be the case in the rest of the document).

For clarity we will list below all the pair's value depending on all possible cases, as listed below. There the numbered cases denote which underlying shock in step 1 was sent first, and alphabetical cases denote the actual underlying and the corresponding sent values, due to incentive compatibility are : (we also organized all the entries below in a summarizing table put at the end of the document, for more readability)

case 1) m_1 is equal to h_A, then m_2 is equal to l_A
case 2) m_1 is equal to l_A, then m_2 is equal to h_A

case a) m is equal to h_A
case b) m is equal to l_A

So the corresponding messages sent by B to A become :

- in the pair (case 1, case a) $Enc_B(Enc_A(m_1)$ - Enc_A(m), $Enc_A(m_2)$ - Enc_A(m)) = (Enc_B((0,1))) = (Enc_B(0),Enc_B(1)).

- in the pair (case 2, case a) $Enc_B(Enc_A(m_1)$ - Enc_A(m), $Enc_A(m_2)$ - Enc_A(m)) = (Enc_B((1,0))) = (Enc_B(1),Enc_B(0)).

- in the pair (case 1, case b) $Enc_B(Enc_A(m_1)$ - Enc_A(m), $Enc_A(m_2)$ - Enc_A(m)) = (Enc_B((1,0))) = (Enc_B(1),Enc_B(0)).

- in the pair (case 2, case b) $Enc_B(Enc_A(m_1)$ - Enc_A(m), $Enc_A(m_2)$ - Enc_A(m)) = (Enc_B((0,1))) = (Enc_B(0),Enc_B(1)).

Again here the value 0 means null, and the value 1 means non-null as will be the case in the rest of the document. Indeed we define a "peculiar" algebra here, which says that ALL non-null numbers will take the same value 1 ie if $Enc_A(m_2)$ - $Enc_A(m) \neq 0$ then $Enc_A(m_2)$ - Enc_A(m) = 1 ; same if $Enc_A(m_1)$ - $Enc_A(m) \neq 0$ then $Enc_A(m_1)$ - Enc_A(m) = 1 . So there are only two different outcomes resulting from the four possible cases 1,2 combined with cases a,b. That will make it easier to conceal what the actual message agent A sent at period 2 is - see summarizing table at the end of this section.

A encrypts back the two messages received from B using his own keys, first the reference $Enc_A(Enc_B(0))$ (the "encrypted null reference"), and the new pair message which value is now encrypted with both B and A's keys, and equal to either $Enc_A(Enc_B((0,1)))$ or $Enc_A(Enc_B((1,0)))$. A sends both the "encrypted null reference" and this new pair message to C. Let's call the new pair message "C's input from A from STEP 2". Indeed, this is linked again to the "information splitting" we described at the start of the protocol, as B doesn't know the order in which h_A and l_A have been sent, but C who knows this order doesn't have the keys to read any of the messages it has to perform homomorphic operations on. Let's note that these messages are also encrypted with A's private key, so even if a malicious B intercepts these messages B still wouldn't be able to read them. Let's also note with the commutative and the distributive properties of the Enc operator, the encrypted vectorized messages (the pairs) become

$$\mathrm{Enc}_A(Enc_B((0,1))) = Enc_B(Enc_A((0,1))) = (Enc_B(Enc_A(0)), Enc_B(Enc_A(1))) =$$

$$(encryptednullreference, Enc_B(Enc_A(1)))$$

and

$$Enc_A(Enc_B((1,0))) = Enc_B(Enc_A((1,0))) = (Enc_B(Enc_A(1)), Enc_B(Enc_A(0))) \tag{30}$$

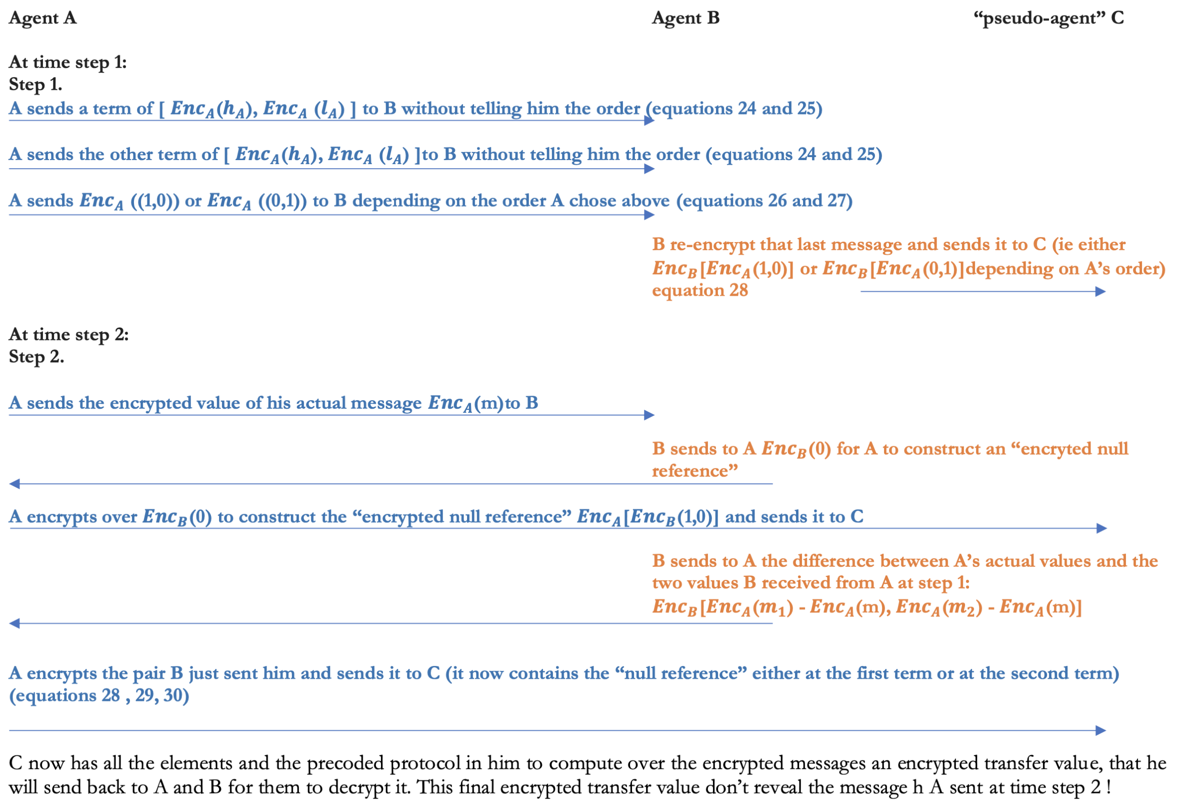
$$\tag{31}$$

$$= (Enc_B(Enc_A(1)), encryptednullreference) \tag{32}$$

.

Let's sum-up all the messages sent between the agents in the following skemata:

Schematic view of the messages being sent in our protocol:



Agent A

Agent B

"pseudo-agent" C

At time step 1:
Step 1.

A sends a term of [$Enc_A(h_A)$, $Enc_A(l_A)$] to B without telling him the order (equations 24 and 25)

A sends the other term of [$Enc_A(h_A)$, $Enc_A(l_A)$]to B without telling him the order (equations 24 and 25)

A sends Enc_A ((1,0)) or Enc_A ((0,1)) to B depending on the order A chose above (equations 26 and 27)

B re-encrypt that last message and sends it to C (ie either $Enc_B[Enc_A(1,0)]$ or $Enc_B[Enc_A(0,1)]$depending on A's order) equation 28

At time step 2:
Step 2.

A sends the encrypted value of his actual message Enc_A(m)to B

B sends to A $Enc_B(0)$ for A to construct an "encryted null reference"

A encrypts over $Enc_B(0)$ to construct the "encrypted null reference" $Enc_A[Enc_B(1,0)]$ and sends it to C

B sends to A the difference between A's actual values and the two values B received from A at step 1:
$Enc_B[Enc_A(m_1)$ - Enc_A(m), $Enc_A(m_2)$ - Enc_A(m)]

A encrypts the pair B just sent him and sends it to C (it now contains the "null reference" either at the first term or at the second term) (equations 28 , 29, 30)

C now has all the elements and the precoded protocol in him to compute over the encrypted messages an encrypted transfer value, that he will send back to A and B for them to decrypt it. This final encrypted transfer value don't reveal the message h A sent at time step 2 !

C is constructed to have "precoded" in it (made sure both by agents A and B as they agree to the scheme) :

- FIRST OPERATION : COMPARE the two terms in the pair of "C's input from A from STEP 2" with the "encrypted null reference", and FIND which term of the pair is equal to the "encrypted null reference" (at least one of the terms should be, since "C's input from A from STEP 2"'s value is either $Enc_A(Enc_B((0,1)))$ or $Enc_A(Enc_B((1,0)))$. And with the notation that $Enc_B(Enc_A((1,0))) = (Enc_B(Enc_A(1)), Enc_B(Enc_A(0))) = (Enc_B(Enc_A(1))$,encrypted null reference)). Keep in memory which term in the pair matched the "encrypted null reference", through a variable $memorized_t$ and one variable NON_$memorized_t$, with $memorized_t$=1 and NON_$memorized_t$=2 if the first term matched the "encrypted null reference" or $memorized_t$=2 and NON_$memorized_t$=1 otherwise. **The variable memorized is in the end keeping track of whether the eventually realized true value was encrypted and sent first, or second, at the very first stage at time step 1 when A sends a message to B, without in itself revealing whether A sent a message for high liquidity need or for low liquidiy need**. Indeed, we built a summary table, Table 1, at the end of this section, of all possible values of $memorized_t$ depending on the case we are in. There we can see for instance that for the high liquidity need value cases (case a, combined with case 1 and case 2) one moves across the two columns with $memorized_t$ going from 1 to 2 depending on if the realized true value (here high liquidity need) was sent first or second by A to B (who then encrypted it using his key and sent to C) at STEP 1. Similarly for the low value cases (case b, combined with case 1 and case 2), $memorized_t$ goes from 2 to 1.

Depending on the result "pseudo agent C" will have to perform the randomization function of Townsend, JME 1988, or execute the transfer corresponding to the low liquidity need (thanks to the randomization in the high liquidity need case, this low liquidity case cannot be distinct with certainty to the result of the randomization in the high liquidity need from A case). **However, because we don't want "pseudo agent C" to know in which case we are even while he is doing it (so that no ones that even manages to "hack" C can reveal A's true liquidity needs) the "pre-programmed" sets of operations has to take place on encrypted space, and has to cover both high liquidity need and low liquidity need cases at once**. For the first of these two requirements we can invoke the homomorphic property of the encryption scheme we have been using all along. For the second requirement we can notice by looking at Table 1 below showing all four cases combining cases 1,2 and cases a,b, that we can use the resulting (at this stage encrypted) 0 and 1 in all four cases as "switches" activating either the randomization function, or the low liquidity need. Indeed the randomization function AND the low liquidity need transfer are coded beforehand IN EACH outcome. So no matter if the actual message sent by agent A at period 2 is the high liquidity need or the low liquidity need both the code doing the randomization and the low liquidity transfer will run. However what changes - the "switches" - are the weights in front of each of these two terms (the randomized number and the fixed low liquidity number). The weights come from C's comparing inputs from A and B, and thus putting an encrypted 0 in front of one of these two terms as weight, and an encrypted 1 in front of the other of these two terms as weight. So in all cases both terms are "computed", but in each case only one of them will actually have a non null weight in the result. Let's write this down more in details :

- SECOND OPERATION : $memorized_t$ from FIRST OPERATION will tell us how each of the two terms in the pair "C's input from B from STEP 1" will be used as the "switches" in the following operation.

Let's note $memorized_t$("C's input from B from STEP 1")= the first term of the pair ("C's input from B from STEP 1") if $memorized_t$=1, else the second term of the pair ("C's input from B from STEP 1") if $memorized_t$=2.

Let's also note NON_$memorized_t$("C's input from B from STEP 1")= the second term of the pair ("C's input from B from STEP 1") if $memorized_t$=1, else NON_$memorized_t$("C's input from B from STEP 1")= the first term of the pair ("C's input from B from STEP 1") if $memorized_t$=2.

I.e., if ("C's input from B from STEP 1")=$Enc_B(Enc_A((0,1)))$ and $memorized_t$=1, then $memorized_t$("C's input from B from STEP 1")=$Enc_B(Enc_A(0))$ and NON_$memorized_t$("C's input from B from STEP 1")=$Enc_B(Enc_A(1))$. Please see all possible cases in Table 1 below.

Notice that in each case one of the two above variables $memorized_t$("C's input from B from STEP 1") and NON_$memorized_t$("C's input from B from STEP 1") will be $Enc_A(Enc_B(0))$, while the other of the variable will be $Enc_A(Enc_B(1))$ (with as a reminder 0 meaning null and 1 meaning non null, and with ALL non null values being equal to 1).

We can now build the following generic set of operations, covering all four cases from the table above with the same sets of instructions :

COMPUTE the output transfer amount resulting from C as the sum between the amount corresponding to the randomization from Townsend, JME 1988, with weight equal to the value of the NOT-YET-DECODED-$memorized_t$("C's input from B from STEP 1"), and the amount corresponding to A's low liquidity need with weight equal to the value of the NOT-YET-DECODED NON_$memorized_t$("C's input from B from STEP 1"). In each of the four case of the table above, only one of the amounts corresponding to the randomization or corresponding to the low value will be non-null, since there is in each case one and only one null weights (even if that null probability is NOT-YET-DECRYPTED - the first of the two requirements to conceal to C and everyone whether randomization was done or not, since even if it's done it's been computed in the encrypted space).

For clarity let's write down the summary table, Table 1, of what messages are sent in each of the four cases above (case 1 or 2, combined with case a or b, where the numbered cases denote which underlying shock in period 1 was sent first, and alphabetical cases done the actual underlying sent at period 2 due to incentive compatibility).

Let's note that in all four cases at item 4) C ALWAYS computes, in all four cases, the generic formula

$memorized_t$(C's input from B from STEP 1).(randomized amount) + NON_$memorized_t$(C's input from B from STEP 1).(low liquidity amount).

Let's also note that homomorphic encryption, along with HE-MPC's distributive and commutative properties is used in 5) of the summary table below, in the equality

$Enc_B(Enc_A(1))$.(randomized amount) + $Enc_B(Enc_A(0))$.(low liquidity need amount)
= $Enc_B(Enc_A($1.(randomized amount) + 0.(low liquidity amount)$))$
= $Enc_B(Enc_A($randomized amount$))$

and in $Enc_B(Enc_A(0))$.(randomized amount) + $Enc_B(Enc_A(1))$.(low liquidity need amount)
= $Enc_B(Enc_A($0.(randomized amount) + 1.(low liquidity amount)$))$
= $Enc_B(Enc_A($low liquidity amount$))$.

We can verify that in all four cases such a homomorphically computed sum between one null term and one non-null term will give the amount corresponding to the specific combination of case 1 or 2, with case a or b that occurred.

So at this point C's output is already a NOT-YET-DECRYPTED numerical value without trace revealing for certain whether the numerical value is the result of a randomization function - the one corresponding to the amount of the transfer that should happen.

Then C sends this NOT-YET-DECRYPTED numerical value back to agents A and B for HE-MPC decryption (see steps 9 to 11 in Figure 2: main steps followed in an HE-MPC computation), so that all three agents and pseudo agent A, B and C knows the value of the transfer in plain numerical form (this value can either be automatically transferred if the contract is linked to a distributed ledger, or be sent to a payment system). Thus, C's intermediate steps, all on encrypted space, and its generic coding that handles at once the high and the low liquidity need case, prevents anyone that could even see C's intermediate results from knowing A's liquidity need.

Table 1: **Summary of messages computed and sent in each of the four cases**

	Case 1: m_1=h_A and m_2=l_A	**Case 2:** m_1=l_A **and** m_2=h_A
Case a: m=h_A	1) C's input from B from STEP 1 =$Enc_B(Enc_A((1,0)))$ 2) C's input from A from STEP 2 =$Enc_A(Enc_B((0,1)))$ 3) C finds $memorized_t$=1, so that NON_$memorized_t$=2 as well 4) The "switches" here take values: $memorized_t$(C's input from B from STEP 1) = $Enc_B(Enc_A(1))$ in front of (randomized amount), and NON_$memorized_t$(C's input from B from STEP 1) = $Enc_B(Enc_A(0))$ in front of (low liquidity need amount) 5) Thus the generic formula takes here the values $Enc_B(Enc_A(1))$.(randomized amount) + $Enc_B(Enc_A(0))$.(low liquidity need amount) = $Enc_B(Enc_A($1.(randomized amount) + 0.(low liquidity need amount)) = $Enc_B(Enc_A($randomized amount))	1) C's input from B from STEP 1 =$Enc_B(Enc_A((0,1)))$ 2) C's input from A from STEP 2 =$Enc_A(Enc_B((1,0)))$ 3) C finds $memorized_t$=2, so that NON_$memorized_t$=1 as well 4) The "switches" here take values: $memorized_t$(C's input from B from STEP 1) = $Enc_B(Enc_A(1))$ in front of (randomized amount), and NON_$memorized_t$(C's input from B from STEP 1) = $Enc_B(Enc_A(0))$ in front of (low liquidity need amount) 5) Thus the generic formula takes here the values $Enc_B(Enc_A(1))$.(randomized amount) + $Enc_B(Enc_A(0))$.(low liquidity need amount) = $Enc_B(Enc_A($1.(randomized amount) + 0.(low liquidity amount)) = $Enc_B(Enc_A($randomized amount))
Case b: m=l_A	1) C's input from B from STEP 1 =$Enc_B(Enc_A((1,0)))$ 2) C's input from A from STEP 2 =$Enc_A(Enc_B((1,0)))$ 3) C finds $memorized_t$=2, so that NON_$memorized_t$=1 as well 4) The generic formula takes here the values $Enc_B(Enc_A(0))$.(randomized amount) + $Enc_B(Enc_A(1))$.(low liquidity need amount) = $Enc_B(Enc_A($0.(randomized amount) + 1.(low liquidity amount)) = $Enc_B(Enc_A($low liquidity need amount))	1) C's input from B from STEP 1 =$Enc_B(Enc_A((0,1)))$ 2) C's input from A from STEP 2 =$Enc_A(Enc_B((0,1)))$ 3) C finds $memorized_t$=1, so that NON_$memorized_t$=2 as well 4) The generic formula takes here the values $Enc_B(Enc_A(0))$.(randomized amount) + $Enc_B(Enc_A(1))$.(low liquidity need amount) = $Enc_B(Enc_A($0.(randomized amount) + 1.(low liquidity amount)) = $Enc_B(Enc_A($low liquidity need amount))

(6) Generalization principles

Let's see what "generalizing" these examples would first mean in the mechanism design framework, before studying how that could be implemented on the technology and encryption side. Let's notice first that generalizing the contracts would mean, in the mechanism design framework, to either increase the number of interacting agents (a), the complexity of the message space (b), or the interaction (computations) to be run between them (c). Let's see for each of them how that would translate to technologically, using the implementation examples we exposed above.

(a) Increasing the number of interacting agents - illustration using the auction example above

The auction example presented in section (4) presents both ways one could tackle an increase in interacting agents - first would be to use the 2 agents case as a basic building block that could iterated making pairwise comparisons over all agents, second would be by adapting the computation function between agents in a way that fit the problem better, globally from the outset. For instance, if one chooses to use the 2 agents case, one can realize that "pseudo agent" C from section (4) is in fact a ranking operator. I.e. presented with any two bids, "pseudo agent" C can figure which is highest without knowing any of the initial bid values. One can thus design a sorting algorithm between N agents' bids using this ranking operator, to find the maximum. One can add in privacy-preserving features if needed - for instance, presenting pairs of bids to be compared randomly to C; letting C encrypt the name of the "winner" of a pairwise comparison so that no agents can know if other agents' bids are higher or lower than his... The engineering gains are in terms of design here since we don't need to alter any of the section (4) set up, while we might be trading off computation time (since we would need to go through all pairs of bids using such a design).

The second approach can improve the number of comparisons, if we increase the complexity of the computation so that instead of taking just 2 agents' bids it can take N agents'. For instance, instead of just doing a substraction of 2 agents' bids to see which is higher, one could do a multiparty addition of all N agents' bids to then compute the difference of all N agents' bids vs the median bid value among the N of them. For the roughly half of these agents that have bids lower than the median, we drop them from the auction, and repeat the process for those agents left in the competition. The multiparty computation complexity is of the same order than the previous case, since we are just doing additions instead of subtractions, and increases linearly with the number of bidders. Please note that one could do that with or without the "pseudo agent C", depending on privacy requirements (as without pseudo agent C then all agents themselves are in charge of running the multiple rounds of multiparty computations, with less and less agents participating in them. But there the values of the successive medians can still be encrypted and kept secret from all, as one only needs to know the sign of the substraction of a bid vs a median. With a "pseudo agent C" then it can be done all by him, so that agents themselves only see the final winner. But that reintroduces the control problem of a trusted third party - in this case pseudo agent C, which can be a server that has been jointly set up by all agents, and then left as a

”deterministic box” on its own, or in a DLT setting a smart contract).

All in all, the trade-offs are linked to the complexity of a potential (re)design process, the complexity of added encryption and multiparty computation steps if needed, and added computation time from repeating encryption and multiparty computation steps if needed, and the privacy considerations one needs to keep.

(b) Increasing the complexity of the message space - illustration using the hybrid contract

Increasing the message space is equivalent, in mathematical terms, to increase the dimensions of that space. For instance, go back to the hybrid's setting above, and assume there are now three different messages (high, medium, low) that could be sent by the borrowing agents instead of the two (high, low) in the previous setting.

There are now two ways one could accomodate such an enlarged message space. The first would be, similarly to the first approach in more agents auction described above, to adapt the existing set up as a basic solver to be repeated several times. Indeed, because there are now 3 potential message values now, we will break these 3 potential values into 2 subproblems containing each only 2 potential values, so that each of these 2 subproblems are EXACTLY the same problem as the case presented above. Therefore we would have

- one pseudo agent (C1) that will ”treat” the low and medium messages - but C1 doesn't know which ones they are doing

- one pseudo agent (C2) that will ”treat” the medium and high messages, but they don't know which ones they are doing

- one pseudo agent (C3) that will ”treat” the low and high messages, but they don't know which ones they are doing

Then, the first pseudo agent is EXACTLY as in our previous hybrid write up, but just with (low, medium). That nodes sends out at the very end: Dec[Enc(actual message value - low)] * Allocation_corresponding_to_low + (1/2) * Dec[Enc(actual message value - medium)]*Allocation_corresponding_to_medium , with the notation that if a subtraction is equal to zero above then Dec[Enc(0)]=1, else then Dec[Enc(non-null value)]=0 (as described in section 5 in the 2 message value case). The second contract does EXACTLY the same with medium and high as it did in the 2 value case, and that node sends out: Dec[Enc(actual message value - high)] * Allocation_corresponding_to_high + (1/2) * Dec[Enc(actual message value - medium)]*Allocation_corresponding_to_medium. Finally we just sum the values sent by both contracts.

The second approach, as described in the auction case, would be to adapt the design of the existing set up so that it can take into account more messages at once. For instance, through the intermediary variables used to ”store” messages in the 2 agents case. In the language of section (5), the *memorized* variable used previously will be a vector instead of the previous scalar values it would take (0 or 1). ie *memorized* is now taking values in (0,0,1), (0,1,0), (1,0,0) for the 3 values of high, medium, or low. Then the SECOND OPERATION performed by pseudo agent C would now be for C to allocate the value [(first term of the memorized vector) times value corresponding to low message + (second term of the memorized vector) times value corresponding to medium message + (third term of the memorized vector) times value corresponding to high message ” (eventually with interaction terms)].

(c) Increasing the interaction (computations) to be run between agents

This now touches upon the maturity of the underlying technologies. If we use HE and MPC as described in this document, then for linear functions increase in complexity of the computation functions are feasible (with associated costs in computation time, see next subsection). Where things are harder are if we introduce non-linear operations like the activation functions in a neural network, on which HE and MPC research is currently showing promising results (see for instance Blatt, Gusev, Polyakov, Rohloff and Vaikuntanathan 2020). If alternative technologies are needed, other homomorphic solutions designed on distributed ledgers (such as Pedersen commitments in Narula, zkLedger 2018) are of interest. Solutions such as trusted executed environments (TEEs) can be used similarly, though they are hardware solutions - so necessarily introducing a discussion on how to maintain them, similar to that around the ”pseudo agent” concept we introduced in this paper.

To conclude on the technical maturity of HE and MPC, one can review and benchmark existing HE-MPC applications and computation times. This isn't the focus of our paper, in which we follow a mechanism design approach in thinking about incentives to send or not messages, and about how these messages can be kept private while still taking part in planning computations. We refer instead to the following selection of literature listing notable implementations. MPC and/or HE have been for instance utilized to develop a method for protecting privacy in large-scale genome-wide association studies (Kamm et al., 2013). This is an analysis of the likelihood of diseases based on private medical information, phenotype (age, gender, height) and genotypes, so that physicians can make tailored recommendations to their patients while all the individual data used in the analysis remains secure. We think of previous research that has used MPC to run a double auction for the Danish sugar beet market (Bogetoft et al., 2009); securely link Estonian education and tax databases (Bogdanov et al., 2016); run a simulation of a decentralized and privacy-preserving local electricity trading market (Abidin et al., 2016); perform an analysis of the gender wage gap in Boston using data from a large set of Boston employers (Lapets et al., 2016); and proof of concept using real data from large firms to show how to securely calculate aggregated measures over sensitive cybersecurity data (Castro el al., 2020).

Though encryption has its limits, in that problems do not scale up easily, the limit in practice is more for the overall number of participants rather than the number of lines of data or interactions in analysis. Overall advancements are promising. For example, the Estonia project ran on 10 million tax records and half a million education records.

(7) Conclusion: Mechanism design solutions as smart contracts (with and without distributed ledgers)

We do not need a planner or trusted third party. This idea should be clear from the two previous examples, auctions without auctioneers and hybrid borrowing/lending insurance schemes with secrets and no third party. To be clear how this generalizes, for a given environment, we solve for the constrained-optimal multi-party arrangement. Then to implement the scheme with our tools, we make the code that generated the solution to the planner problem available for participating agents to see line by line, so they can validate that the code does what it is supposed to do. This contract node now replaces the planner entirely. This initial validation process can include multiple potential simulations as if being implemented in real time, following the time line which follows. Agents get convinced it achieves their desired purpose, the reason for contracting, and that information is concealed.

The contract can be hashed and signed into an immutable record that agents have entered into it. No denials later. Next, each agent deposits owned assets or liquidity into an escrow account. This could be an escrow account with a trusted third party such as a commercial bank or an escrow account that is a digital asset on a ledger with programmed conditions for use. The messages are sent internally, as described above.

Similar constructions allow delegated portfolio management without trust, consultation with public oracles, and implementation of solutions which mitigate or potentially eliminate bank runs through automated state contingent gateways dependent on recorded immutable history.

To reiterate we do not need distributed ledgers for the mechanism design part of the problem. Indeed, there is a bit of a disconnect in the different uses of the word trust between computer science and economics literatures. Morris and Shin (1996) come to the essence of the problem by modeling the incentives of participants to follow the prescribed protocol in the Byzantine Generals problem. This is a classic and seemingly simple problem of coordinating a successful attack when the enemy may be prepared, in which case an attack even if coordinated will not be successful, and only one general is informed of this state. Generals can send messages back and forth, but messages are noisy, that is, may not arrive albeit with a low probability. Strategic maximizing behavior in an explicit information game, without commitment, is inconsistent with intuitive prescribed protocols, as agents start to second guess each other as in a Bayesian sequential equilibria. An alternative counter-intuitive protocol with commitment and less communication, one that prevents the second general from replying with a validating message, achieves the optimum, ironically.

The larger point is that we cannot be sure that the validators on distributed ledgers will blindly follow the protocol. Instead we should consider strategic behavior. The Morris and Shin example is particularly damning as it is in the group's interest to coordinate. The opposite tactic is taken in implementation of multi-party mechanism design problems, as protocols take into account strategic behavior and private information and thus are constrained-optimal and incentive-compatible from the get go. Apart from faulty computers, there is no need for validation protocols. HE and MPC can be used to design contracts as objects traded, to design data base infrastructure, to design auctions and matching algorithms, and to design and implement appropriate regulation to deal with privacy and monopolistic concerns. Notably, encryption and multi-agent mechanism design can help the BIS as tools to help foster better coordination across central banks. Indeed, schemes such as the ones presented in this outline can be similarly applied in multilateral processes to ensure the fair and exact execution of chosen computations or of chosen multiagent contracts, without directly accessing nor revealing to anyone but their originators any of their sensitive information.