14.129
Spring, 2025
Lecture 10

Designs of Financial Infrastructure Utilizing Encryption

Auctions, Hybrid Credit and Insurance

Robert M. Townsend

Elizabeth & James Killian Professor of Economics, MIT

Massachusetts Institute of Technology

Applications

- Auction
- Hybrid Credit and Insurance
- Market Implementation

First application: Auctions

Auctions

- 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.
  - 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.

Auctions “Without An Auctioneer"

- First scheme, Auction-Without-Auctioneer Scheme with HE and MPC
  - Each bidder uses his own server with encrypted messages back and forth to other bidders.
  - no third party server nor contract node
  - Each buyer sends his encrypted bid (with public/private key pairs, as in HE) to the others
  - In the MPC part, they send results (encrypted messages) to each other.
- Second scheme, with contract node
  - All communication goes through a third party server as the contract node.
  - Communication of encrypted messages is with this server (can’t see private values), is doing in the code what the agents were doing above.

Example, Step 1, Cont.

- 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 (Brakershi, 2012; Fan and Vercauteren, 2012, BFV scheme) key generation protocol, which outputs a public-key (pk)/secret-key (sk) pair of the form

$$(pk, sk) = \big((a, \delta), (s, e)\big) \quad (1)$$

    - a and δ: two uniformly random samples over the ring $R_q = \frac{Z_q[x]}{f(x)}$, all participants use the same pseuorandom sample a from R_q; $f(x)$: some degree-n polynomial; s: secret key; e: an error term from a discrete Gaussian.
  - Each party i then generates the following key pair:

$$(pk_i, sk_i) = \big((a, \delta), (s_i, e_i)\big) \quad (2)$$

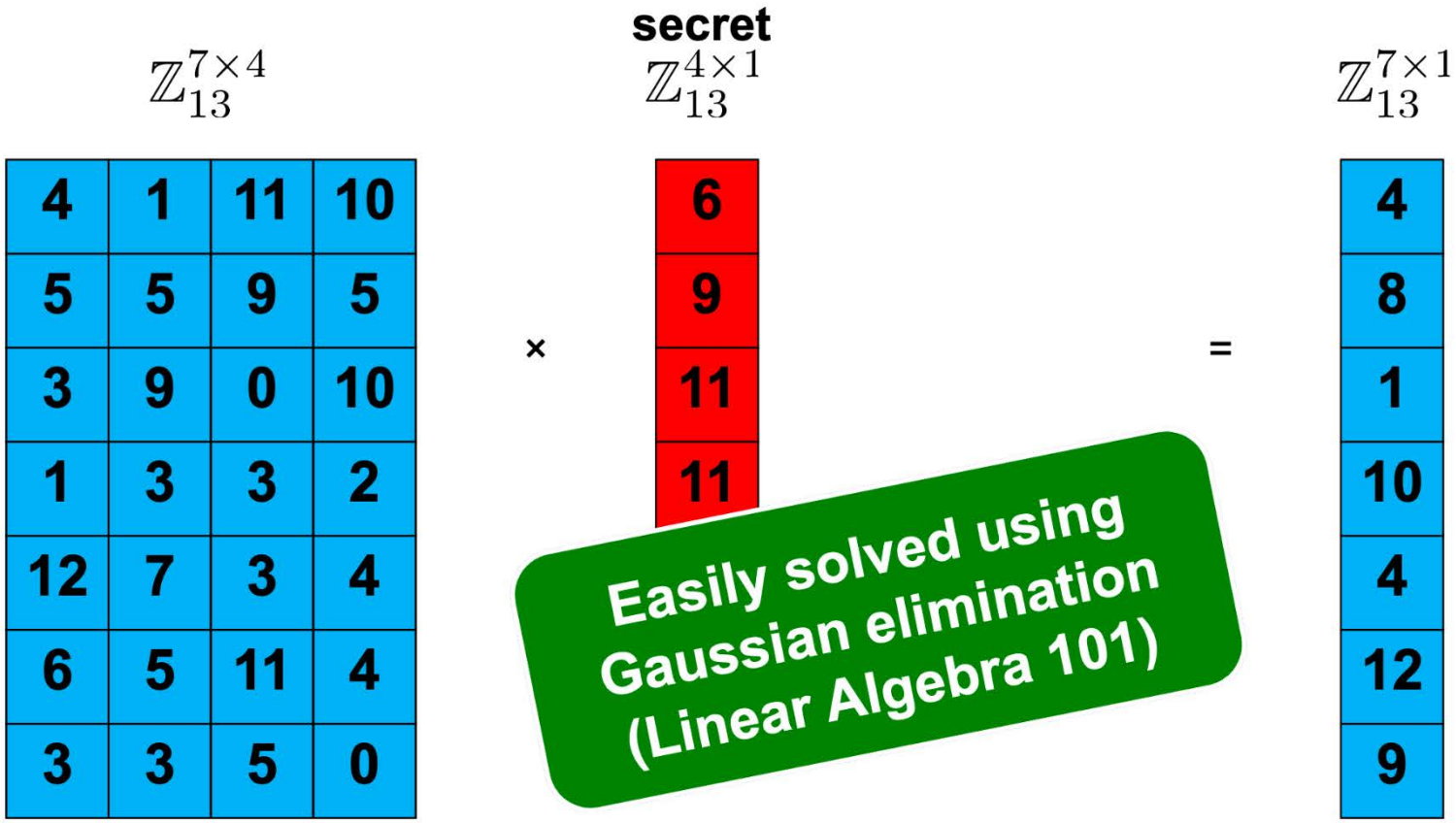
  - So agent A and B have the following key pair, respectively:

$$(pk_a, sk_a) = \big((a, \delta), (s_a, e_a)\big) \quad (3)$$

$$(pk_b, sk_b) = \big((a, \delta), (s_b, e_b)\big) \quad (4)$$

What exactly is the error term - motivation

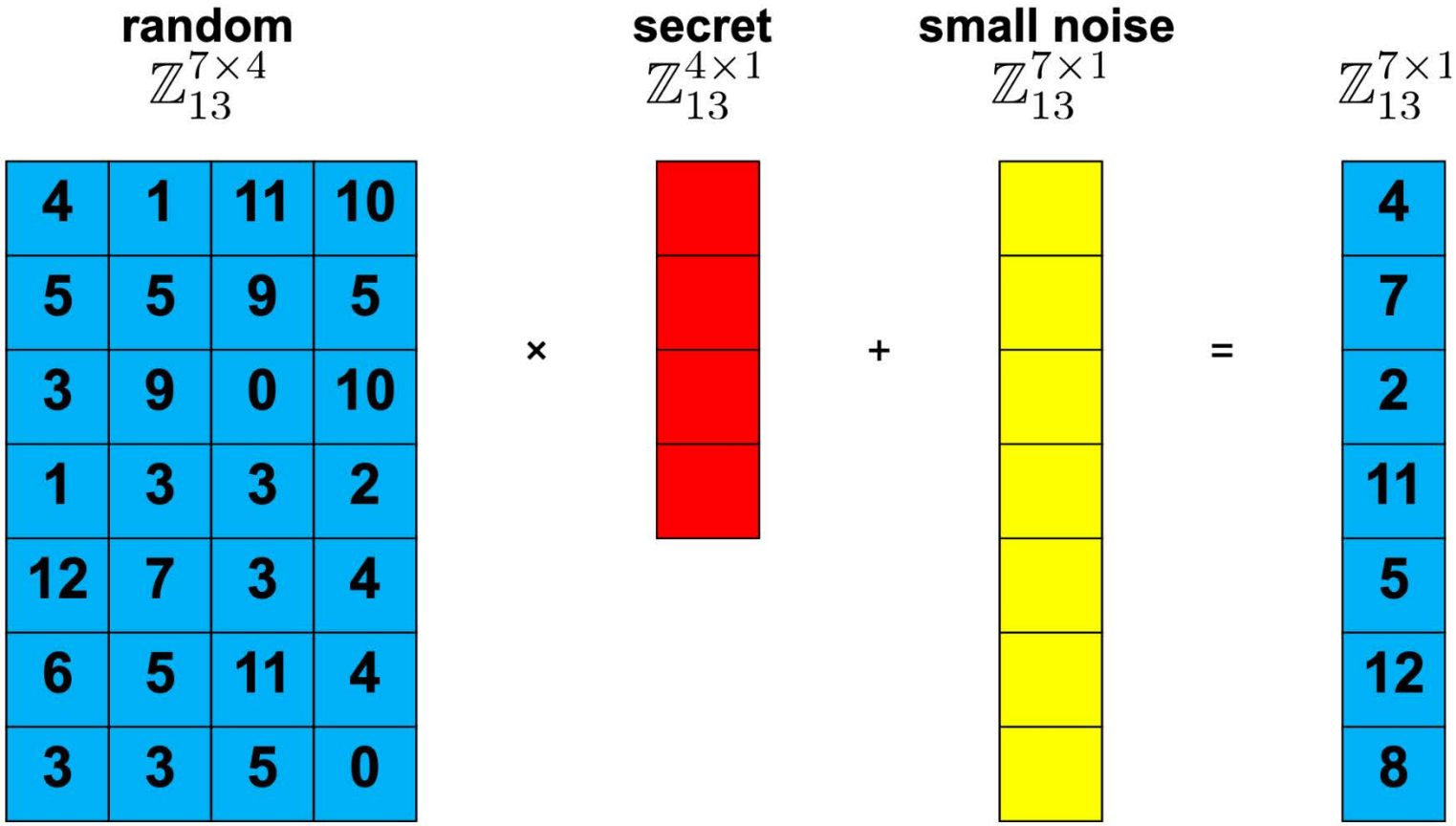
Solving systems of linear equations



Linear system problem: given **blue**, find **red**

What exactly is the error term - motivation

Learning with errors problem



Search LWE problem: given **blue**, find **red**

First Scheme Example, Step 1, Cont.

- **Multiparty encryption of each agent's bid**
  - Now that each agent has its unique secret key, while sharing a public key, they can start using these to encrypt their bids.
    - Let's say A starts (it would be the same if it's B first) : A sends to B its bid ct_a, in particular with an unknown intercept MPC
    - $ct_a = a \cdot s_a + \delta \cdot m_a + e_a$ (5)
    - B cannot guess (from the security proof of the BFV scheme) the value of A's bid m_a.
    - B sends similarly his encrypted bid m_b to A, ct_b: MPC
    - $ct_b = a \cdot s_b + \delta \cdot m_b + e_b$ (6)
    - δ is a public scaling factor to prevent the small error terms from corrupting the message as $\delta . m\_i >>$ e_i

First Scheme Example, 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. Specifically,

  For Agent A, add in on top of ct_b : $a\, s_a + e_a$ MPC (7)

  For Agent B, add in on top of ct_a : $a\, s_b + e_b$ MPC. (8)

- So now agent A has the new ciphertext $ct_b^{'}$, by writing (9-11)

  $e_a + e_b = e,$ (9)

  $s_a + s_b = s$ (10)

  $ct_b^{'} == a \cdot s + \delta \cdot m_b + e$ (11)

- And agent B has the new ciphertext $ct_a^{'}$

  $ct_a^{'} == a \cdot s + \delta \cdot m_a + e$ (12)

- Both of them resend the ciphertext to each other, then both can do the operation

  $ct_a^{'} - ct_b^{'} = ct_{Final} = \delta \cdot (m_a - m_b)$ MPC (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. But think of it as simply rank order op
  - 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.

Example of Second Scheme

Second Scheme Example, Using the 2 Agents Case to Introduce the Concept of "Pseudo Agent" Nodes

- How can a third party decrypt a result without having access to agents' secret keys ? (HE)

- Agent A and B still have the following key pair, respectively:

$$(pk_a, sk_a) = \big((a, \delta), (s_a, e_a)\big) \quad (14)$$

$$(pk_b, sk_b) = \big((a, \delta), (s_b, e_b)\big) \quad (15)$$

- Now in step 1, both A and B send their ciphertexts to C, a pseudo agent, and C also "asks" both A and B to send the nn_a and nn_b but these are still not decipherable

$$nn_a = a \cdot s_a + e_a \quad (16)$$

$$nn_b = a \cdot s_b + e_b \quad (17)$$

- 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) respectively, while allowing C to decrypt ct_c:

$ct_c = ct_a - ct_b = a \cdot s + \delta(m_a - m_b) + e$ using definitions (5), (6). (18)

Correction: Eq (18) should be primes

- Indeed, we have :

$$nn_a + nn_b = a \cdot s + e \quad (19)$$

- so that from (18)

$$ct_c - (nn_a + nn_b) = \delta(m_a - m_b) \quad (18) - (19) = (20)$$

Notes: Pseudo node C has equations 16 and 17 to add in the private keys. Eq (19) is not needed, but see next slide.

Generalized MPC

(1) Each agent individually generates its own key pair, where each key pair contains a public encryption key and a private decryption key.
(2) All agents submit their public keys to the server.
(3) The server combines all agents' public keys into a single joint/shared public key.
(4) This new joint/shared public key is distributed from the server to all agents.
(5) Each agent encrypts its private data using this new joint/shared public key, generating a ciphertext (an encrypted block of data).
(6) Each agent sends the ciphertext of its private data to the server. This ciphertext completely hides the agent's data.
(7) The server runs computations on all the encrypted data, producing an encrypted result of the computation.
(8) The server sends the encrypted result back to each agent.
(9) Each agent uses the private key they generated in Step 1 to partially decrypt the answer.
(10) Each agent sends this partially decrypted answer back to the server. Note that the number of agents without which partial decryption from all remaining agents will still give a completely encrypted result, is a parameter that can be chosen beforehand by the mechanism designer.
(11) The server combines the results of all the partial decryptions it receives from agents to produce the decrypted result that is then shared with all agents.

main steps followed in an HE-MPC computation

Increasing the Number of Interacting Agents- Illustration Using the Auction Example Above

- The auction example presented above presents two 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 be iterated making pairwise comparisons over all agents.
    - For instance, if one chooses to use the 2 agents case, one can realize that “pseudo agent” C is in fact a ranking operator, figuring out 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
  - **second** would be by adapting the computation function between agents in a way as to fit the problem better. Specifically, increase the complexity of the computation so that instead of taking just 2 agents’ bids it can take N agents’ bids
    - For instance, one could use some computer science ”tricks” to sort faster among bids.
      - Eg partition all the bids in subgroup, compute averages for each subgroup, and take out all bids lower than any of these averages. And repeat the process for those agents left in the competition. The multiparty computation complexity is of the same order than the previous case, 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.

Second application: *hybrid borrowing and lending with insurance*

liquidity problems – a reminder

- Money transfer organizations run **short of liquidity**
  - In Kenya, agents for Safaricom are frequently running out of at 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.
  - In New York markets, broker dealers as agents 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.

Mechanism Design w/ Planner: intuition

- Viewing agents' specific liquidity shortages as idiosyncratic shocks
  - Ex ante insurance is good
  - But revealing those shocks over time too quickly can damage beneficial trades: ex ante insurance possibilities are lost once the ex post adverse event has occurred.
- Casting application as a mechanism design problem (feature concealment), featuring a example:
  - Two agents (agent a and agent b) having utility functions subject to privately observed preference, two periods, and a planner
  - Agent a in the first period can be patient or urgent to consume, sending private knowledge (message) to the planner.
  - Likewise for agent b in the second period
  - The "planner" sees the messages, but makes sure the other agent does not.
  - Set incentives to tell the truth in second period
  - The information-constrained optimal allocation is that lying in the second period generates a very bad outcome for the sender with positive probability.

Model, Townsend (1988)

- Two agents a and b and two time periods t = 0 and t = 1
- In each period t, both agents receive a deterministic endowment e_t^i with $e_t = e_t^a + e_t^b$ known by both agents
- preferences are given by the objective function

$$U^i(c_0^i, \theta_0^i) + \beta V^i(c_1^i, \theta_1^i)$$

  - c_0^i, c_1^i is path for agent i 's consumption
  - $(\theta_0^a, \theta_1^a, \theta_0^b, \theta_1^b)$ comes from a finite set Θ which represents a consumption shock prole for both agents
- θ_t^i is revealed only to agent i at time t and is therefore private information
- Each element of $(\theta_0^a, \theta_1^a, \theta_0^b, \theta_1^b) \in \Theta$ occurs with probability $p(\theta_0^a, \theta_1^a, \theta_0^b, \theta_1^b)$

Mediation, mutual fund Townsend (1988)

- Assume that there exists an insurance provider as mediator in this economy
  - Collect information from the two agents and decide on how to split the endowment in each period based on the agents reports
- Construct an allocation rule that maximizes a weighted sum of the two agents ex ante expected utility
- Direct Revelation: Let the message space be the set of all possible θ_0^i, at $t = 0$ and the set of all possible (θ_0^i, θ_1^i), and $t = 1$ for each agent i
- Revelation Principle: Any equilibrium allocation of an arbitrary mechanism in a truthful equilibrium can be implemented as a truthful equilibrium of a direct revelation mechanism

Example: Optimal Learning as Optimal Scrambling, Townsend (1988)

- The mediator may choose to "scramble" messages
- Mediator may choose a distribution over consumption that, even given both agents reporting truthfully, one or both of the agents may not be able to discern with certainty the $t = 0$ type of the other agent given the observed c_0

Example

- Set θ_0^b and θ_1^a to be constant
  - only agent a is reporting at $t = 0$ and agent b is reporting at $t = 1$
- Preferences are given by

$$U^a(c_0^a, \theta_0^a) = V^a(c_1^a, \theta_1^a). \qquad (c_t^a)^{\theta_t^a}$$

with $\theta_0^a \in \{0.2\ , 0.9\}, \theta_1^a = 0.30$ for sure

and

$$U^b\left(c_0^b, \theta_0^b\right) = V^b\left(c_1^b, \theta_1^b\right) \qquad \left(c_t^b\right)^{\theta_t^b}$$

with $\theta_0^b = 0.9\ for\ sure\ ,\ \theta_1^b \in \{0.2\ , 0.9\}$

Example cont

- Parameters θ_0^a and θ_1^b have the following joint distribution

$$\left(\theta_0^a, \theta_1^b\right) = \begin{cases} (0.2, 0.9) & with\ prob.\ 0.3 \\ (0.9, 0.9) & with\ prob.\ 0.2 \\ (0.2, 0.2) & with\ prob.\ 0.1 \\ (0.9, 0.2) & with\ prob.\ 0.4 \end{cases}$$

- $e_t^a = e_t^b = 5\ for\ t = 0, 1, \beta = .95, and\ \lambda^a = \lambda^b = 0.5$

Example: Fully revealed communication

- Consider the fully revealed communication solution:

Full 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)	$\pi(c_1, \theta_0^a,$
0.2	(1.75, 8.25)	1.0	(0.2, 0.2)	(2, 5, 7.5)	1.0
			(0.2, 0.9)	(2.5, 7.5)	1.0
0.9	(0.0, 10.0)	0.1042598	(0.9, 0.2)	(3.75, 6.25)	1.0
	(3.25, 6.75)	0.8957402			
			(0.9, 0.9)	(1.0, 9.0)	0.89189814
				(10.0, 0.0)	0.10810186

- b already knows agent $a^{'}s$ type and thus scrambling is unhelpful, limited insurance for b

Communication Concealed: Optimal Scrambling

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)	$\pi(c_1, \theta$
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			
	(1.75, 8.25)	0.0339681	(0.9, 0.2)	(3.75, 6.25)	1.0
	(3.25, 6.75)	0.8544384			
			(0.9, 0.9)	(1.0, 9.0)	0.861060201
				(10.0, 0.0)	0.138397942

- The allocation $(c_0^a, c_0^b) = (1.75, 8.25)$ occurs with positive probability for either report of agent a
  - When $(c_0^a, c_0^b) = (1.75, 8.25)$ is observed, agent b remains uncertain of the type agent a reported in $t = 0$

Example: Optimal Learning as Optimal Scrambling, Townsend (1988)

- Presence of the scrambling in the private information case keeps agent b uninformed of agent $a^{'}s$ type making her more hesitant to lie when $(c_0^a, c_0^b) = (1.75, 8.25)$ is observed and $\theta_1^b = 0.2$ is her type
  - She may be in the $\theta_1^a = 0.9$ case which has a positive probability of a very bad outcome of $c_1^b = 0$

Implementation Notes For the Hybrid Borrowing Lending Case without Central Planner

- Let's use the same setting as the previous auction between two agents: agents A and B, and "pseudo agent" C, which helps to conceal new elements.
- To summarize,
  - agents A and B negotiate a risk pooling contract carried out over two periods , time step 2.
  - There is also an initial contract setting at time step 1, and at the end of which agents A and B put their assets as endowments into escrow.
  - In period 2, 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, so that B is not to be able to know with certainty what message A sent in the first place).
  - Here the "central planner" is replaced by the "pseudo agent" C.

Optimal scrambling with MPC, Schematic view

Agent A (pre-contract) Agent B "pseudo-agent" C

(first term is m1 in subsequent notation)

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)

(other term is m2 in subsequent notation)

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

(real time , during which contract is executed)

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" $Enc_A[Enc_B($] 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:
$Enc_B[Enc_A(m_1) - Enc_A(\text{m}), Enc_A(m_2) - Enc_A(\text{m})]$

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 !

Period1: preplay set up and communication.

- Recalling the notation of key pairs and encryption from last section:

$$Enc_A(h_A) = a \cdot s_a + \delta \cdot h_A + e_a \quad (21)$$

$$Enc_A(l_A) = a \cdot s_a + \delta \cdot l_A + e_a \quad (22)$$

- **Assuming that $\delta = 1$**, so that the Enc operator is commutative here, i.e., with message m

$$Enc_B(Enc_A(m)) = Enc_A(Enc_B(m)) = a \cdot s_a + a \cdot s_b + m + e_a + e_b \quad (23)$$

- This is true for any message m. In practice schemes such as BFV can be made commutative, even if the mathematics behind become a bit "heavier".
- From there on let's call $Enc_A(m_1)$ the first message value agent B received, and $Enc_A(m_2)$ the second message agent B received. If A sent $Enc_A(h_A)$ first and $Enc_A(l_A)$ second, then

$$Enc_A(m_1) = Enc_A(h_A) \quad (24)$$

$$Enc_A(m_2) = Enc_A(l_A) \quad (25)$$

- and vice and versa for the other. But B doesn't know which was sent first, high or low.
- Agent A also sends to B, if A sent to B $Enc_A(h_A)$ first and $Enc_A(l_A)$ second:

$$Enc_A((1,0)) = (a \cdot s_a + 1 + e_a;\ a \cdot s_a + 0 + e_a) \quad (26)$$

Here the value 0 means null, and the value 1 means non-null,

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

$$Enc_A((0,1)) = (a \cdot s_a + 0 + e_a;\ a \cdot s_a + 1 + e_a) \quad (27)$$

- As encrypted messages, these are uninterpretable by B. B then encrypts this message with his own key pairs, in effect adding, and B sends that to C.

$$Enc_B(Enc_A(1,0)) = (a \cdot s_a + a \cdot s_b + 1 + e_a + e_b\ , a \cdot s_a + a \cdot s_b + 0 + e_a + e_b) \quad (28)$$

if A sent to him $Enc_A(h_A)$ first and $Enc_A(l_A)$ second, vice versa. Let's call that message

)m B from STEP 1".

Period 2. Execution in real time

- Let m be the message A sends at t=2.
- B 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, $Enc_A\big(Enc_B(0)\big)$: the "encrypted null reference" but its usage below is more 0 means consistent with what actually happened.:

- Indeed B also sends to A the encrypted pair $\Big(Enc_B\big(Enc_A(m_1) - Enc_A(m)\big),\ \ Enc_B\big(Enc_A(m_2) - Enc_A(m)\big)\Big) = Enc_B\big[Enc_A(m_1) - Enc_A(m)\big), Enc_A(m_2) - Enc_A(m)]$ from distributive property. Note that this is a vector (a pair), and this schemes can be made commutative.

- For $Enc_B\big[Enc_A(m_1) - Enc_A(m)\big), Enc_A(m_2) - Enc_A(m)],$ let
  - 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 m2 is equal to h_A
  - case a) m is equal to h_A
  - case b) m is equal to l_A
- So, there are four situations:

1. (case 1, case a) $Enc_B(Enc_A(m_1) - Enc_A(m), Enc_A(m_2) - Enc_A(m)) = \ Enc_B(0,1) = (Enc_B(0), Enc_B(1))$
2. (case 2, case a) $Enc_B\big(Enc_A(m_1) - Enc_A(m), Enc_A(m_2) - Enc_A(m)\big) = Enc_B(1,0) = (Enc_B(1), Enc_B(0))$
3. (case 1, case b) $Enc_B(Enc_A(m_1) - Enc_A(m), Enc_A(m_2) - Enc_A(m)) = \ Enc_B(1,0) = (Enc_B(1), Enc_B(0))$
4. (case 2, case b) $Enc_B(Enc_A(m_1) - Enc_A(m), Enc_A(m_2) - Enc_A(m)) = \ Enc_B(0,1) = (Enc_B(0), Enc_B(1))$

- There are only two different outcomes resulting from the four possible cases. The two outcome allows C to send the right allocation even without knowing actual m (next slide).Either the first message was the actual value, as in (0,1) or the second as in (1,0)- next slide

Period 2. At Time Step 2, Cont.

- We want C (which doesn't have the keys to read any of the messages it has to perform homomorphic operations on) to act as if actual message m is high or low, **while everything is encrypted , both with A's private key, so even if a malicious B intercepts, B still wouldn' t be able to read**

- So A encrypts back the two messages received from B using his own keys, and send the messages to C:
  - first the reference $Enc_A(Enc_B(0))$: **encrypted null reference**;
  - and second the new pair message either $Enc_A(Enc_B(0,1))$ or $Enc_A(Enc_B(1,0))$. Let's call the new pair message **"C's input from A from STEP 2".**

- Let's also note with the commutative and the distributive properties of the Enc operator, the encrypted vectorized messages (the pairs) become

$$Enc_A(Enc_B(0,1)) = Enc_B(EncA\ (0,1)) = \left(Enc_B(Enc_A(0)), Enc_B(Enc_A(1))\right) = (encrypted\ null\ reference, EncB(EncA(1)$$

and

$$Enc_A(Enc_B(1,0)) = Enc_B(EncA\ (1,0)) = \left(Enc_B(Enc_A(1)), Enc_B(Enc_A(0))\right) = \left(EncB(EncA(1)), encrypted\ null\ reference\right)$$

Period 2. At Time Step 3, First Operation

C is constructed to have "precoded" in it:

- **First Operation : Compare** the two terms in the pair of "**C's input from A from STEP 2**", i.e., either $Enc_A(Enc_B(0,1))$ or $Enc_A(Enc_B(1,0))$ " **with the "encrypted null reference"** i.e., $Enc_A\big(Enc_B(0)\big)$, and **FIND which term of the pair is equal to the "encrypted null reference"**
- Specifically, for $Enc_A\big(Enc_B(1,0)\big) = Enc_B\big(EncA\ (1,0)\big) = \Big(Enc_B\big(Enc_A(1)\big), Enc_B\big(Enc_A(0)\big)\Big)$

  $= \big(EncB\big(EncA(1)\big), \boldsymbol{encrypted\ null\ reference}\big)$.

  - Keep the position of the term "encrypted null reference" in memory, through variables $\boldsymbol{memorized_t}$ and $\boldsymbol{NON\_memorized_t}$.
  - Here, since the second term matched the "encrypted null reference", so $memorized_t = 2$ and $NON\_memorized_t = 1$. (*Vice versa, for* $Enc_A\big(Enc_B(0,1)\big), memorized_t = 1$ *and* $NON\_memorized_t = 2$)
- The variable $\boldsymbol{memorized_t}$ is in the end keeping **track of whether the eventually realized true value m was encrypted and sent first, or second**, at the period 1 at time step 1 when A sends a message m to B, without in itself revealing whether A sent a message for high liquidity need or for low liquidity need.

Period 2. At Time Step 3, Second Operation: Compute

- Let's note, **if $\boldsymbol{memorized_t = 1}$** (from first operation),
  then $\boldsymbol{memorized_t}$ **("C's input from B from STEP 1")** = the **first term** of the pair **("C's input from B from STEP 1")**
  and $\boldsymbol{NON\_memorized_t}$ **("C's input from B from STEP 1")= the second term of the pair**
- Else, if $memorized_t = 2$, $memorized_t(\text{"}C's\ input\ from\ B\ from\ STEP\ 1\text{"}) = the\ second\ term\ of\ the\ pair\ (\text{"}C's\ input\ from\ B\ from\ STEP\ 1\text{"})$, and $NON\_memorized_t(\text{"}C's\ input\ from\ B\ from\ STEP\ 1\text{"}) = the\ second\ term\ of\ the\ pair.$
- Here, $memorized_t(\text{"}C's\ input\ from\ B\ from\ STEP\ 1\text{"})$ and $NON\_memorized_t(\text{"}C's\ input\ from\ B\ from\ STEP\ 1\text{"})$ are **"switches".**

i.e., if $(\text{"}C's\ input\ from\ B\ from\ STEP\ 1\text{"}) = Enc_B\big(Enc_A(0,1)\big) = Enc_B\big(\big(Enc_A(0), Enc_A(1)\big)$

$$and\ memorizedt = 1,$$

$$then$$

$$memorized_t(\text{"C's input from B from STEP 1"}) = Enc_B\big(Enc_A(0)\big)$$

$$and$$

$$NON\_memorized_t(\text{"C's}\ input\ from\ B\ from\ STEP\ 1\text{"}) = \mathrm{Enc_B}(\mathrm{Enc_A}(1)).$$

- Notice that in each case one of the two above $memorized_t(\text{"C's input from B from STEP 1"})$ and $NON\_memorized_t(\text{"C}'s\ input\ from\ B\ from\ STEP\ 1\text{"})$ will be $Enc_B\big(Enc_A(0)\big)$, while $\mathrm{Enc_B}(\mathrm{Enc_A}(1)$ (here also 0 meaning null and 1 meaning non null).

Period 2. At Time Step 3, Pseudo Agent C always Computes the Generic Formula

- **Principle: use encrypted "switches"** to activate either the randomization function, or the low liquidity need. 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 both run. Let's write this down more in details:

- **We implement these "switches" through the pair of encrypted variables,** $memorized_t$ **and** $NON\_memorized_t$

- **So now the generic function, always computed, is:**

$$memorized_t(C's\ input\ from\ B\ from\ STEP\ 1) \cdot (randomized\ amount)$$
$$+ NON\_memorized_t(C's\ input\ from\ B\ from\ STEP\ 1) \cdot (low\ liquidity\ amount).$$

- At this point C's output is already a NOT-YET-DECRYPTED numerical value without t 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.

Period 2. At Time Step 3, Pseudo Agent C always Computes the Generic Formula

- Now use the "switches" to activate either the randomization function, or the low liquidity need. 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. Let's write this down more in details:

$$memorized_t(C's\ input\ from\ B\ from\ STEP\ 1)\cdot(randomized\ amount)$$
$$+\ NON\_memorized_t(C's\ input\ from\ B\ from\ STEP\ 1)\cdot(low\ liquidity\ amount).$$

- Let's also note that homomorphic encryption, along with HE-MPC's distributive and commutative properties, in the equality

$$Enc_B(Enc_A(1))\cdot(randomized\ amount)\ +\ Enc_B(Enc_A(0))\cdot(low\ liquidity\ need\ amount)$$
$$= Enc_B(Enc_A\big(1\cdot(randomized\ amount)+\ 0\cdot(low\ liquidity\ amount)\big)$$
$$= Enc_B(Enc_A(randomized\ amount))$$

and

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

- 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 [illegible] corresponding to the amount of the transfer that should happen.

Period 2. At Time Step 3, Pseudo Agent C's Last Operation

- Then pseudo agent 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 A, B and C agents know the value of the transfer in plain numerical form.

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

- 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.

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_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) $\models$ $Enc_B(Enc_A(1))$ 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))

Increasing the Complexity of the Message Space – general principle

- Increasing the message space: for instance, back in the hybrid' s setting above, and assume there are now three different messages **(high, medium, low)** that could be sent by agent a.

- A first "natural" way to accommodate for such a richer message space:
  - if one recalls the "switches" - put three now instead of two

- - in the two switches case one switch activated the low need value , one switch activated the high need randomization. Here the three switch case one switch activates high, one activates intermediary, one activates low.

  - The *memorized* variable will be a vector instead of the previous scalar values, (0 or 1): *memorized* now takes values in (0,0,1), (0,1,0), (1,0,0) for the 3 values

  - Then the second operation performed by pseudo agent C: C allocate the value [(first term of the memorized vector)*value corresponding to low message + (second term of the memorized vector)*value corresponding to medium message + (third term of the memorized vector)*value corresponding to high message ((eventually with interaction terms)].

- This process however can't be parallelized if there are the message space is complex. In that case, one can instead work with the "comparator" of slide 35 ("1st operation of period 2, time step 3")

Increasing the Complexity of the Message Space – parallelization

- looking at the three different messages **(high, medium, low)** that could be sent by agent a **in any order ((high, medium, low), (medium, low, high)…** in time step 1 (pre contract):
  - if one recalls the ”comparator" of slide 35
- - in 2 message case a simple difference between *actual value* vs *first value received* will determine fully if first message received was high (then second low), and vice versa, as there are only two possibilities
  - For 3 messages one needs to both determine what first message received was, what second message received was, before the value of the third message received is fully determinate
  - With thus break this more complex problem in subproblem of still size two (possible message values), with then our same old design used several times (can be in parallel!)
  - Therefore we would have - one pseudo agent (C1) that will compare the first two received message with **low** and **medium** values; one pseudo agent (C2) that will compare the first two received message with **medium** and **high** ; one pseudo agent (C3) that will compare the first two received message with **low** and **high**

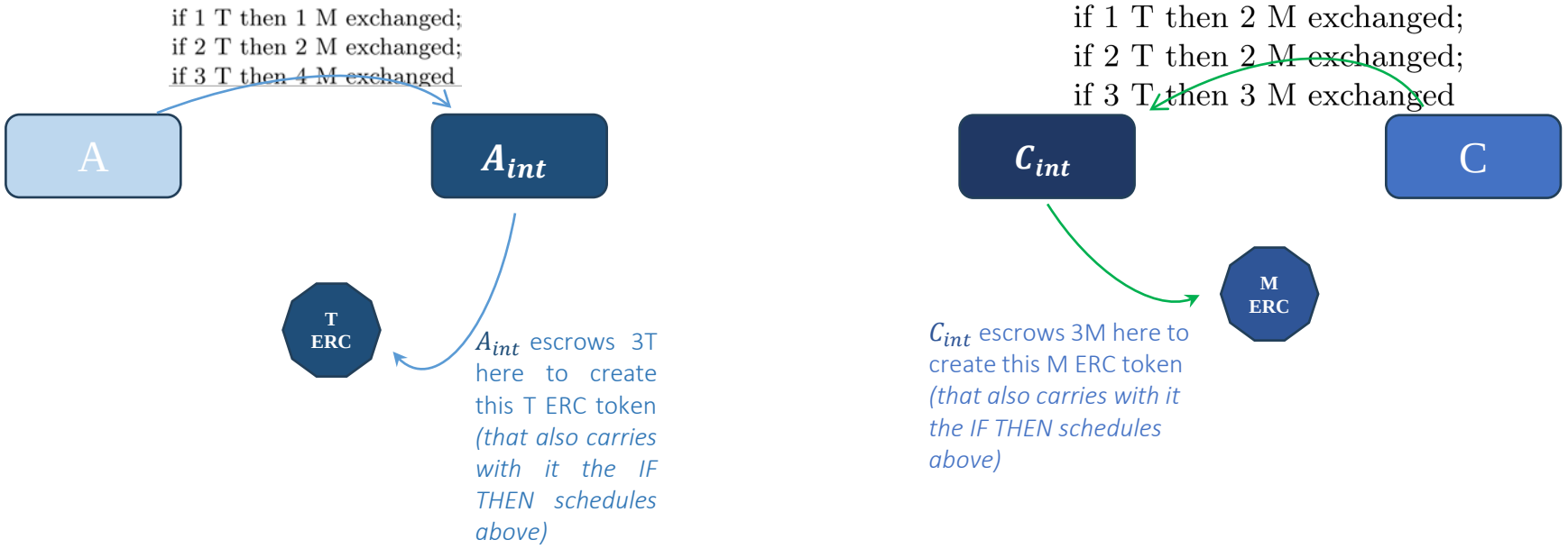
- **Then one of these pseudo node (any of them) could at the end aggregate the allocation resulting from all of these subproblems**
  - *Ie from C1 : (1/2)*{Dec[Enc (actual message value-low)]* Allocation_corresponding_to_low*
  - *+Dec[Enc(actual message value-medium)]* Allocation_corresponding_to_medium]}*
  - *From C2 : (1/2)*{Dec[Enc (actual message-medium)]* Allocation_corresponding_to_medium*
  - *+Dec[Enc(actual message value-high)]* Allocation_corresponding_to_high]}*
- ***with notation that is a subtraction is equal to zero above then Dec[Enc(0)]=1, else then Dec[Enc(non-null value)]=0;***

Order Book Matching

Contract With Clients

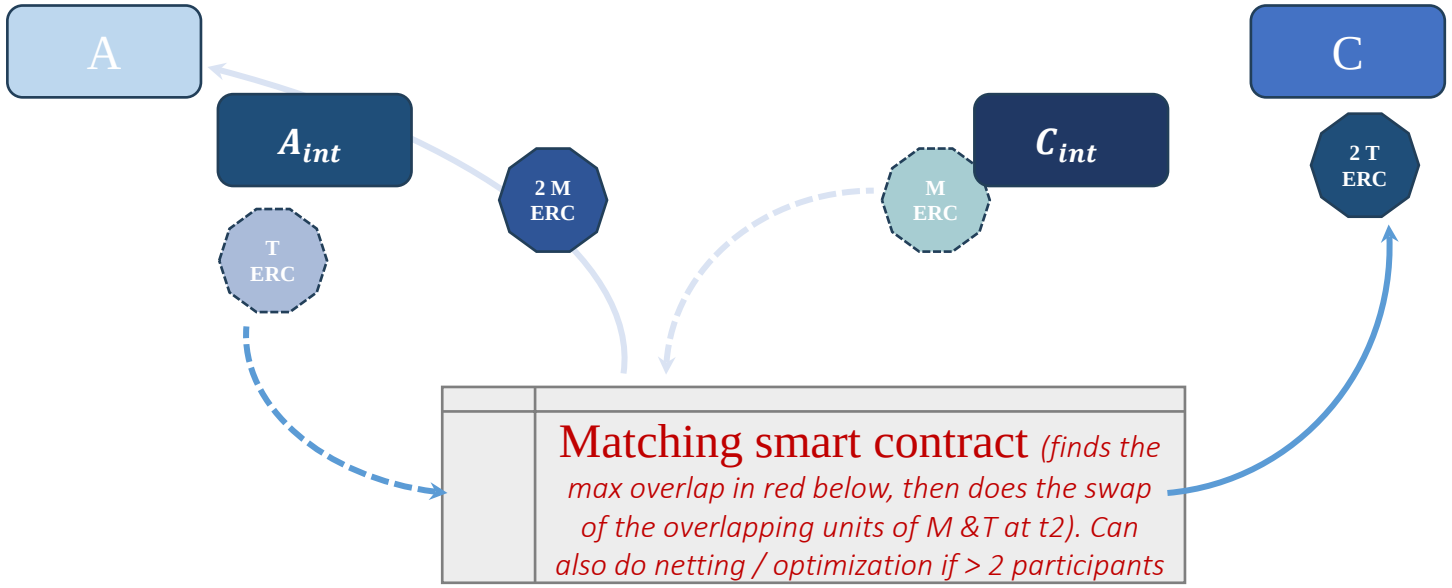
At t0 (negotiation time pre-entering messages & assets into smart contracts):

- *At t0* Clients & BDs agree to different schedules & scenario; the BDs commit to honor these & lock these in a custom ERC token each *(each token carries in its data field these different schedules & scenario, and the BDs put in escrow in these ERC the max value of each's schedule)*



Matching Smart Contract

- *At t1* BDs submit these ERC tokens in a matching smart contract which finds the maximum overlapping trade *(here for instance 2T for 2M).*



at t1: matching contract finds the max overlap of both schedules. If no overlap then the central bank can be asked to bridge the difference

if 1 T then 1 M exchanged;
if 2 T then 2 M exchanged;
if 3 T then 4 M exchanged

if 1 T then 2 M exchanged;
if 2 T then 2 M exchanged;
if 3 T then 3 M exchanged

Encryption and Implementation on the Blockchain

- Bid schedules can be encrypted and comparison done in FHE space
- Chatzigiannis, Townsend, Aronoff, Zhang, Minaei, Raghuraman, and Bhat “SoK: Fully-homomorphic encryption in smart contracts”