KU LEUVEN

Private Liquidity Matching using MPC

Shahla Atapoor, Nigel P. Smart and Younes Talibi Alaoui

nigel.smart@kuleuven.be

ia.cr/2021/475

Outline:

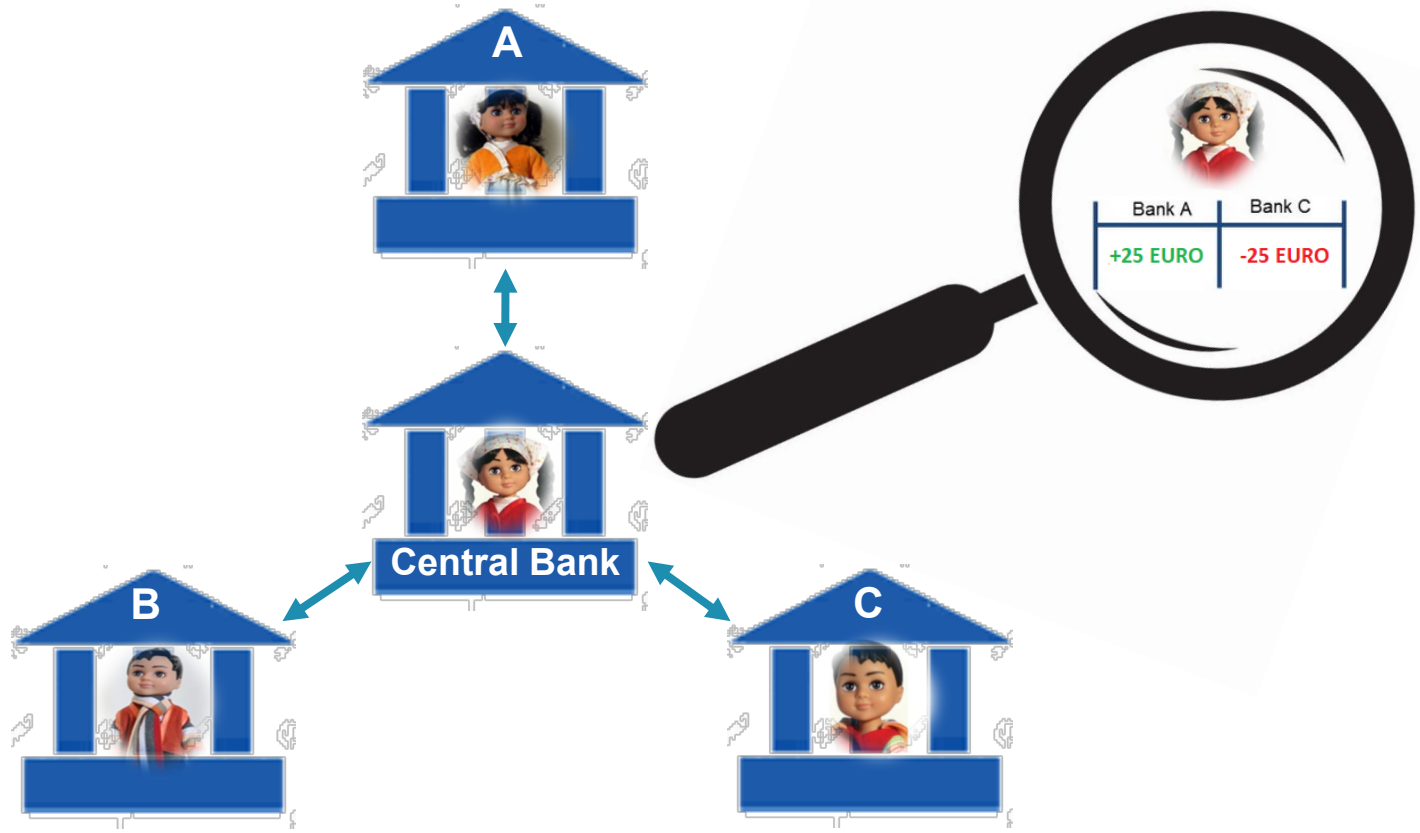
- Introduction and Background
  - RTGS systems
  - Liquidity of banks
  - GridLock Resolution Problem (GRP)
- Problem statement
  - Decentralized RTGS system
- Our Contribution
  - Main idea
  - Algorithms
    - Sources Open & Destinations Open (SODO)
    - Sources Open & Destinations Secret (SODS)
    - Sources Secret & Destinations Secret (SSDS)
  - Performance
    - Runtimes of three different algorithms
    - Simulation result

Payment Systems: Real Time Gross Settlement Systems (RTGS)

RTGS:

This is the back bone of modern banking. Transactions are settled individually and immediately at real time if the source has enough liquidity.

- Real Time = Immediately
- Gross = Individually
- Settlement = Debiting money from the source account and crediting to the destination account.

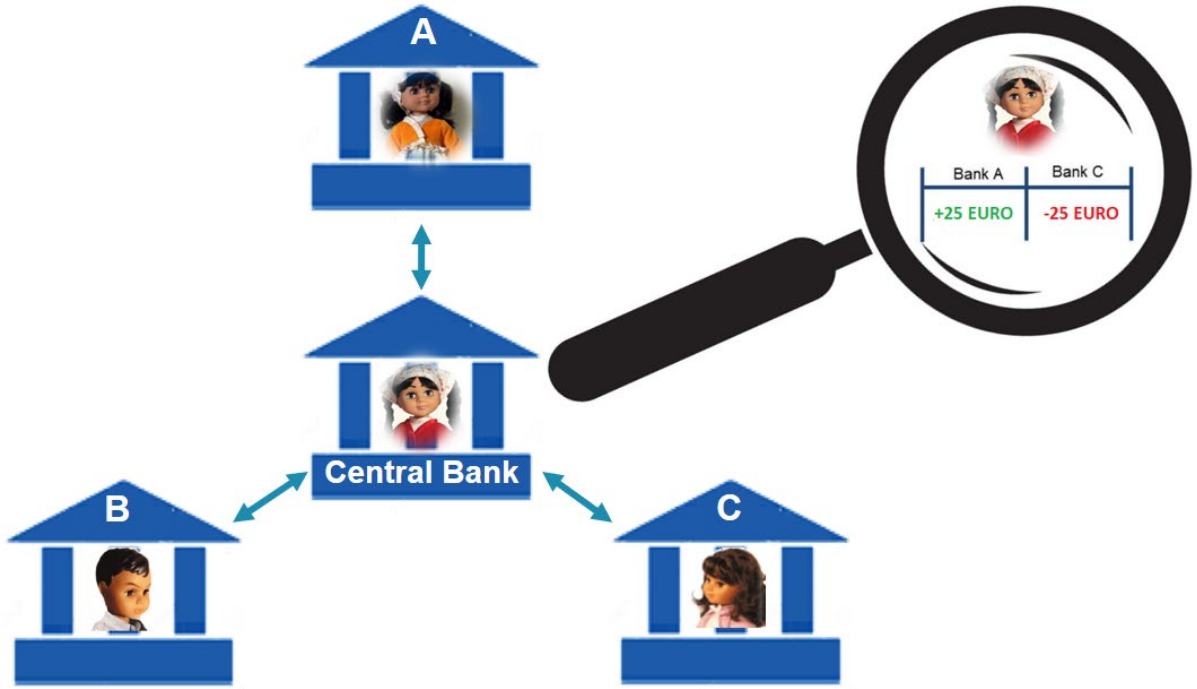


Payment Systems: Real Time Gross Settlement Systems (RTGS)

Upside: RTGS systems mitigate the effects caused by a failing bank not being able to settle its payments, as opposed to netting systems where the settlement happens at the end of the day.

Downside: problems with some of liquidity cases.

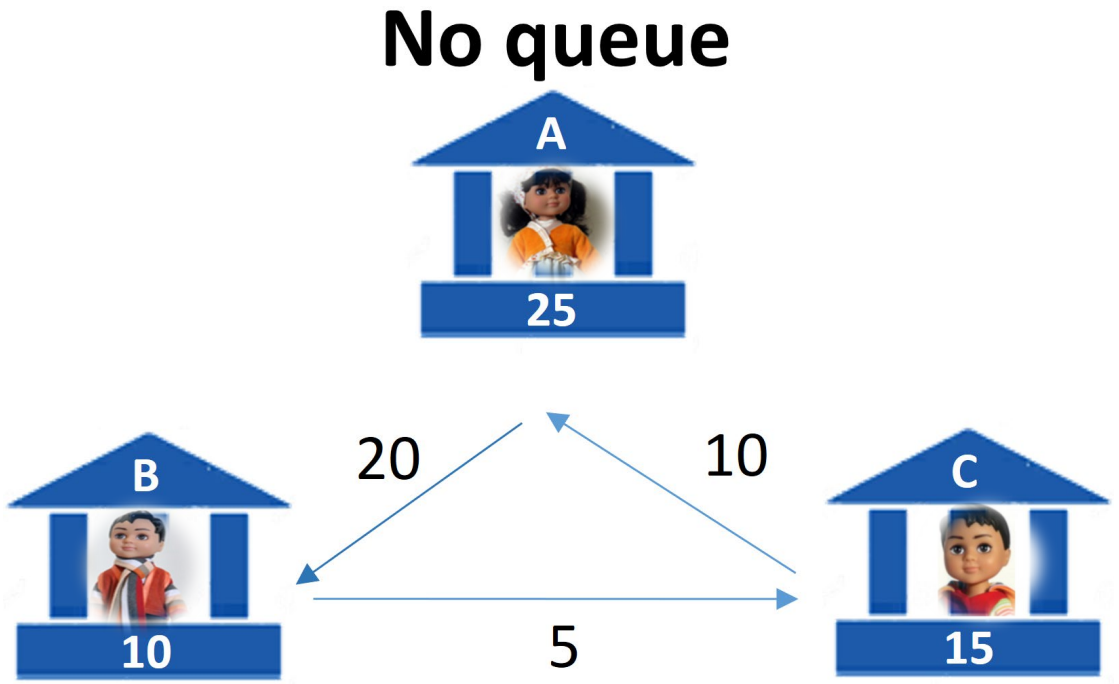
- No queue ☺
- GridLock 😐
- DeadLock ☹



Payment Systems: Liquidity of Banks-No queue ☺

❑ **Examples:** No queue in Payment Systems.

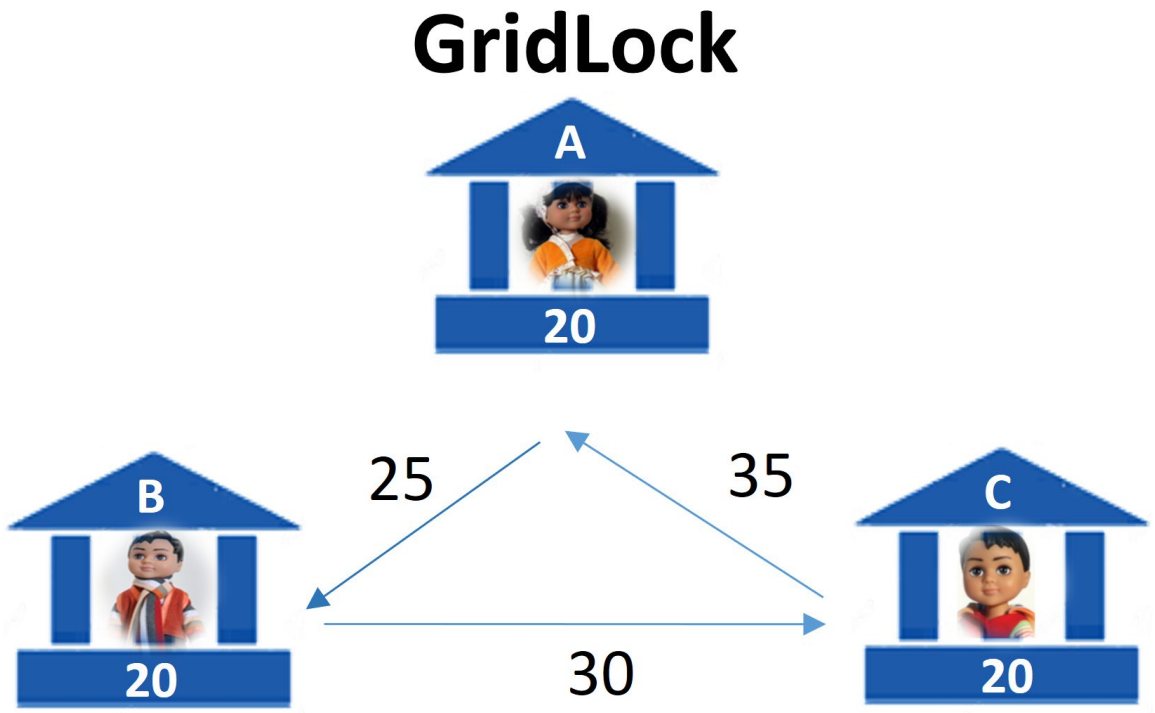
- Transactions execute immediately
- Each individual bank has enough liquidity to complete the transactions



Payment Systems: Liquidity of Banks- GridLock 😐

❑ **Examples:** GridLocks in Payment Systems.

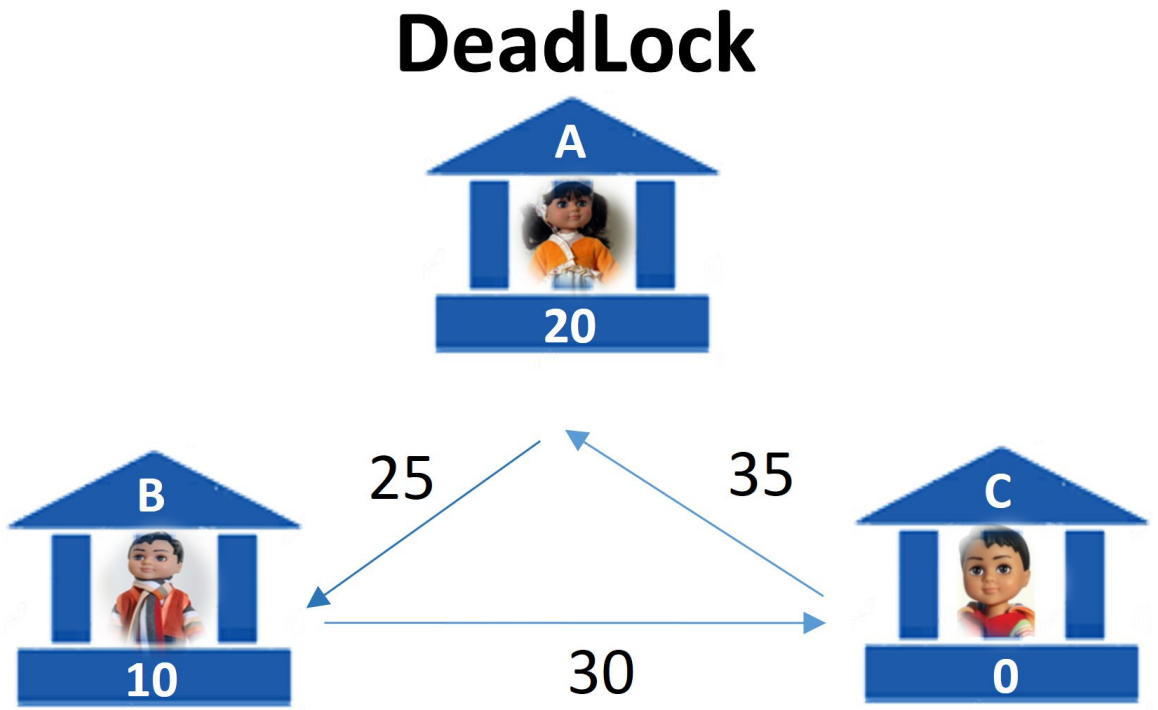
- Transactions could complete, if we looked at the global view of what is owed to who.
- The system does have enough liquidity.



Payment Systems: Liquidity of Banks- DeadLock ☹

❑ **Examples:** DeadLocks in Payment Systems.

- There is not enough liquidity in the system to complete the transactions.



Payment Systems: Real Time Gross Settlement Systems (RTGS)

If enough liquidity in the system transactions clear instantly.

What to do in case transactions are pending:

- Option 1: The parties involved need to inject more liquidity into the system.
- Option 2: Liquidity Saving Mechanisms (LSM) can be employed such as Multilateral Netting:
  - Consists of simultaneously offsetting multiple transactions.
  - Permits to unblock some of the pending transactions if net positions of the respective sources are positive.
  - This problem is called the Gridlock Resolution Problem..
  - This process is generally carried out by a central entity

Question: Can we remove the central entity for Option 2?

Payment Systems: GridLock Resolution Problem (GRP)

GridLock Resolution's Problem (GRP) ≡ A discrete optimization problem

Aims to maximize the number of transactions to be settled subject to:

- The balances of the participants should not be negative.
- The priority of settling the payments preferred by banks should be preserved.

Payment Systems: Solution to the GRP [Bech and Soramäki, 2001]

- **Solution:** Namely, maximizing the number of payments, to be picked from the queue of each bank.
  - Under the assumption that the transactions are strictly ordered.

1) Include all queued payments in the solution.
2) Calculate balances for all the banks,
   - If there is at least one negative balance then execute step 3.
   - If all the balances are positive then stop.
3) Take all banks with a negative balance and remove their last transaction from the queue of the solution. Repeat step 2 until no such banks remain

Outline:

- Introduction and Background
  - RTGS systems
  - Liquidity of banks
  - GridLock Resolution Problem (GRP)
- Problem statement
  - Decentralized RTGS system
- Our Contribution
  - Main idea
  - Algorithms
    - Sources Open & Destinations Open (SODO)
    - Sources Open & Destinations Secret (SODS)
    - Sources Secret & Destinations Secret (SSDS)
  - Performance
    - Runtimes of three differnt algorithms
    - Simulation result

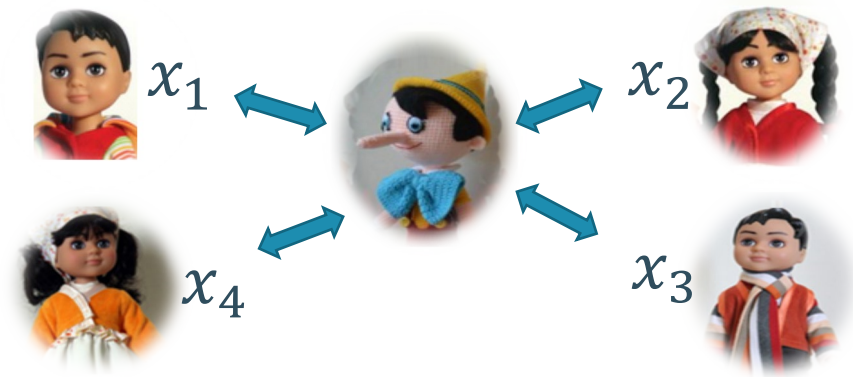
European Central Bank (ECB) Systems: Motivation

There is an interest in removing the need for the central authority:

- For example could the role of the central bank be provided by Distributed Ledger Technologies?

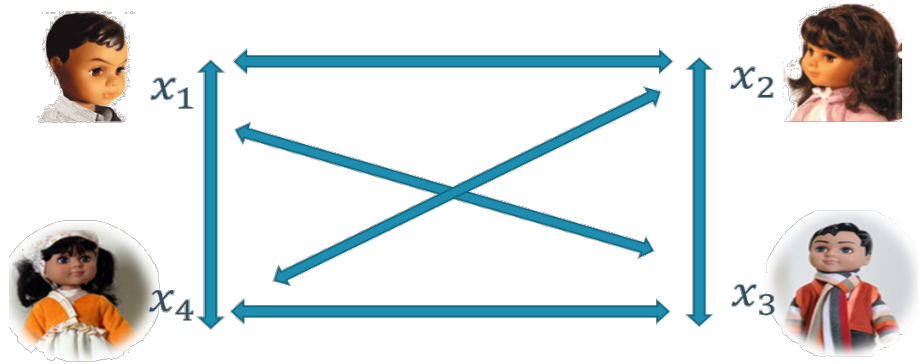
This would be a problem as to solve the GRP we need banks to exchange all transaction information:

- Bank's A and B would need to disclose their bilateral transactions to bank C.
- This is not in their business interests to do so.



Solution

- Decentralize the RTGS system by allowing the GRP to be solved using a secure distributed algorithm.



Payment Systems: Requirements of Decentralized RTGS

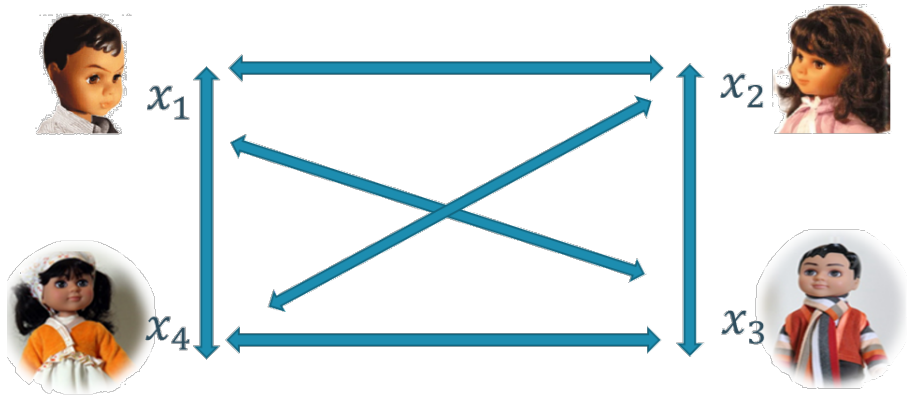
A secure decentralized RTGS system will have three main requirements:

Correctness: while settling a transaction, the amount debited from the source is the same as the amount transferred to the destination.

Fairness: The LSM process implemented should not favor a participant over the others.

Security: Transaction information between two entities should not leak to a third.

Our solution looks at three different definitions of information in the third security requirement.



Outline:

- Introduction and Background
  - RTGS systems
  - Liquidity of banks
- Problem statement
  - Decentralized RTGS system
- Our Contribution
  - Main idea
  - Algorithms
    - Sources Open & Destinations Open (SODO)
    - Sources Open & Destinations Secret (SODS)
    - Sources Secret & Destinations Secret (SSDS)
  - Performance
    - Runtimes of three different algorithms
    - Simulation result

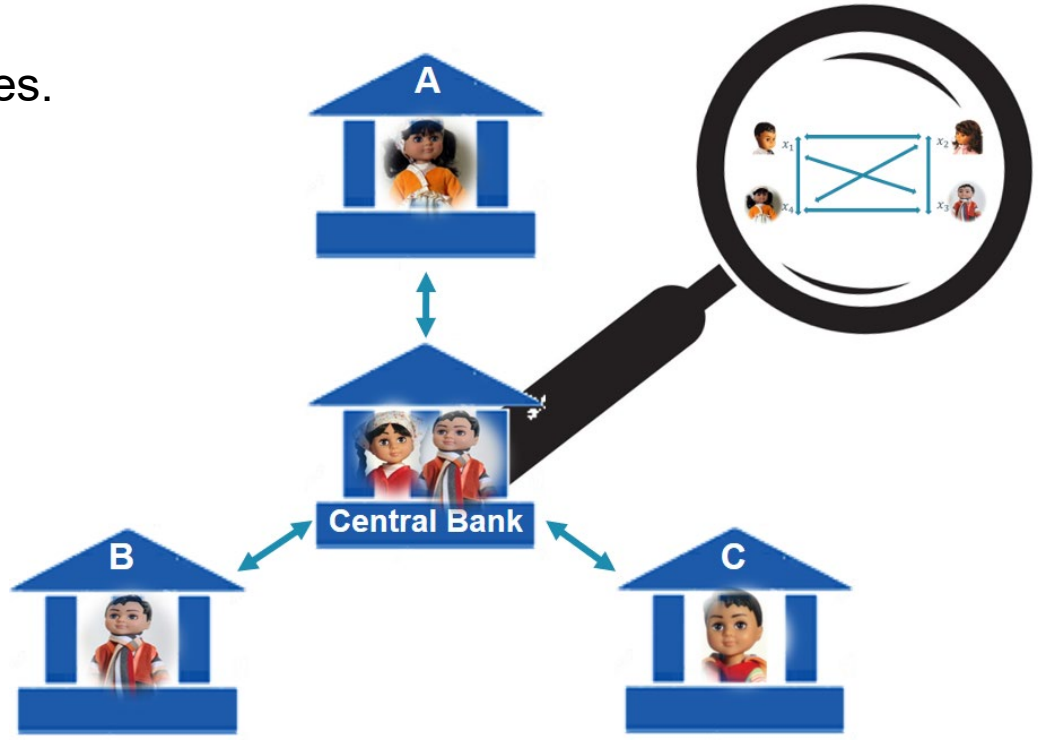
Our Contribution: An Efficient Multi-Party Computation (MPC) Protocol

An MPC based solution to perform the liquidity optimization for decentralized RTGS systems.

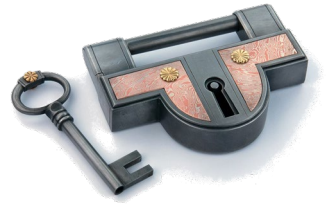
Task of managing the RTGS system assigned to a set of entities.

The payments instructions and balances will remain hidden as long as those entities do not collude.

The entities will be capable of obliviously running a multilateral netting process.



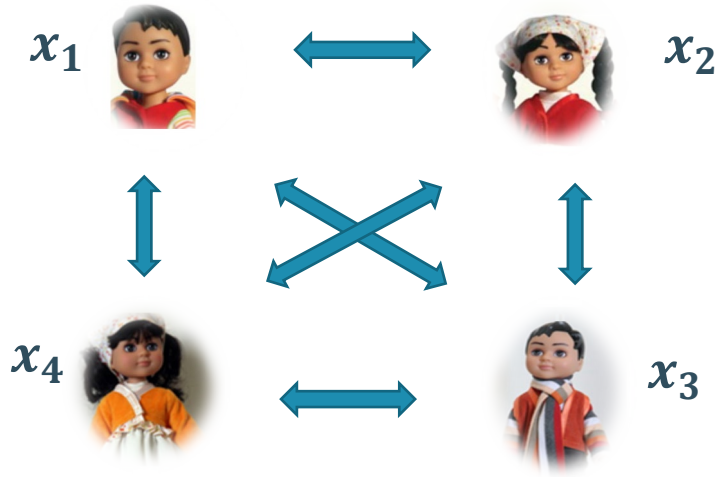
Preliminary: Multi-Party Computation (MPC)



Let F(.) be a function represented with a binary/an arithmetic circuit.

Definition: Parties are computing F(.) on shared data

$$y = F(x_1, x_2, x_3, x_4).$$



Security Requirements:

- Security and integrity of computation.
- Parties learn nothing from the other parties inputs but the correct output.

Our Setting: We Use LSSS

Linear Secret Sharing Schemes (LSSS):

- Secret is shared linearly, e.g. $x=x_1+x_2+\cdots+x_n$.
- We write [x].
- Linear operations (e.g. addition) can be done locally.
- Non-linear operations (e.g. multiplication) requires communication.

Our Setting: We Use LSSS

We are using MPC based Shamir Secret Sharing with Abort.

Secret [x] is shared linearly using an (n,t)-threshold, e.g. Shamir:

- Value x is shared by finding a polynomial f of degree t with f(0) = x
- Share to player i is the value $x_i = f(i)$
- Abort: if a party cheats, the other parties detect this and abort the computation (since we select t< n/2)

Introduction: MPC Protocol: Offline Phase

We are using two phases: Offline and Online phase

Offline Phase

Beaver triples: [a], [b], [c] with c = a*b.

Beaver triples are generated before the protocol starts.

This “offline” phase is function independent.

Introduction: MPC Protocol : Online Phase

To add:

Additions are free

Given two secrets x= f(0), y = g(0)

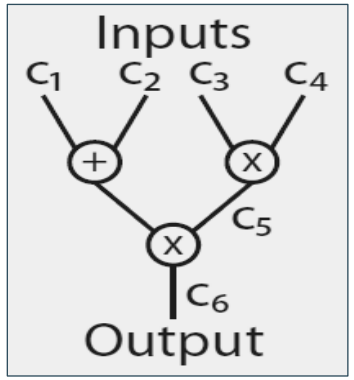
Their sum is shared by the polynomial h(x) = f(x)+g(x).

- z = x + y = h(0) = f(0) + g(0)

Share of z is given by

- $z_i = h(i) = f(i) + g(i) = x_i + y_i$

This allows us to compute any linear function.



Introduction: MPC Protocol: Online Phase

To Multiply :

Supose you have Beaver triples: [a], [b], [c] with a = b* c

- Open r = [x] – [a]
- Open s = [y] – [b]
- Compute [z] = [x] * [y] = r * s + s * [x] + r * [y] + [c]

This last operation is a linear operation, and hence can be done using the previous slides techniques.

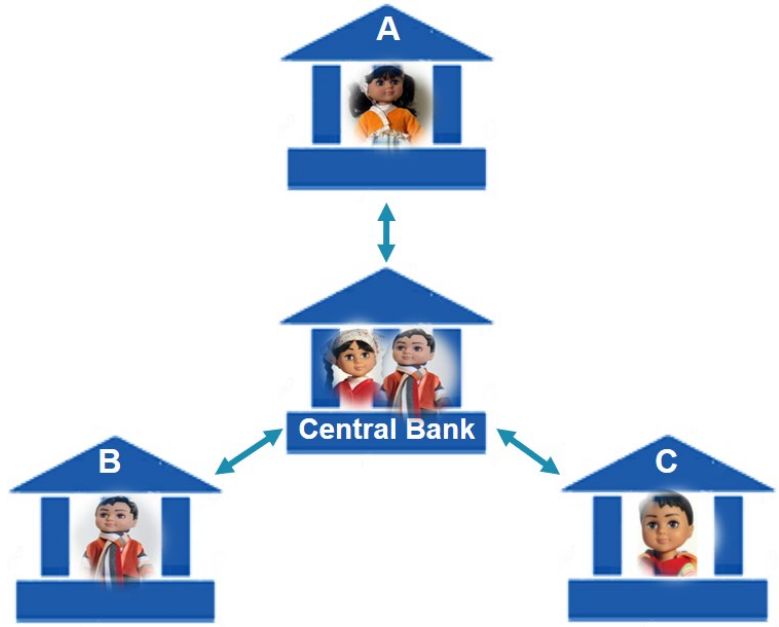
MPC in ECB Systems: Pipeline

Three Versions of MPC Algorithms:

The algorithms for solving GRP

- SODO: Sources Open, Destinations Open and amount of transactions are secret
- SODS: Sources Open, Destinations Secret and amount of transactions are secret
- SSDS: Sources Secret, Destinations Secret and amount of transactions are secret

Source = Name of source bank
Destination = Name of destination bank
Amount = The value being transferred
Bank Balances always remain secret



Our Contribution: Solution to the GRP with MPC- Initial Challenge of SODO

1) Include all queued payments in the solution.
2) Calculate balances for all the banks,
   - If there is at least one negative balances then execute step 3.
   - If all the balances are positive then stop.
3) Choose the bank with the negative balance and remove from the last payment in queue for this bank from the solution. Repeat step 2.

We want to compute the balances after all transactions have completed $[B_U^i]$ given input balances $[B^i]$.

Let $[x_t]$ denote a variable which indicates whether a transactions in the queue should be included.

Initially $[x_t] = 1$ for all transactions in the queue.

We then execute:

- For all i in [1,…,n] do
  - $[S^i] = \Sigma\ [a] * [x_t]$ where sum is over all transactions t = (s,[a],r) with source i.
  - $[R^i] = \Sigma\ [a] * [x_t]$ where sum is over all transactions t = (s,[a],r) with destination i.
  - $[B_U^i] = [B^i] - [S^i] + [R^i]$.

Our Contribution: Solution to the GRP with MPC- Initial Challenge of SODS

1) Include all queued payments in the solution.
2) Calculate balances for all the banks,
   - If there is at least one negative balances then execute step 3.
   - If all the balances are positive then stop.
3) Choose the bank with the negative balance and remove from the last payment in queue for this bank from the solution. Repeat step 2.

If the destination is secret we need to alter the sum for $[R_i]$.

Using a naïve ORAM implementation we demux the index i via a demux array $[C_{t,i}]$, where t is a transaction and i is the index for the destination.

We then execute:

- For all i in [1,…,n] do
  - $[S^i] = \Sigma\ [a] * [x_t]$ where sum is over all transactions t =(s, [a], [r]) with source i.
  - $[R^i] = \Sigma\ [a] * [x_t] * [C_{t,i}]$ where sum is over all transactions t = (s, [a], [r])and all i.
  - $[B_U{}^i] = [B^i] - [S^i] + [R^i]$.

Our Contribution: Solution to the GRP with MPC- Initial Challenge of SSDS

1) Include all queued payments in the solution.
2) Calculate balances for all the banks,
   - If there is at least one negative balances then execute step 3.
   - If all the balances are positive then stop.
3) Choose the bank with the negative balance and remove from the last payment in queue for this bank from the solution. Repeat step 2.

If the source is secret we need to alter the sum for $[S_i]$.

Use another demux array $[W_{t,i}]$, where t is a transaction and i is the index for the source.

We then execute:

- For all i in [1,…,n] do
  - $[S^i] = \Sigma\ [a] * [x_t] * [W_{t,i}]$ where sum is over all transactions t =([s], [a], [r]) with source i.
  - $[R^i] = \Sigma\ [a] * [x_t] * [C_{t,i}]$ where sum is over all transactions t = ([s], [a], [r])and all i.
  - $[B_U{}^i] = [B^i] - [S^i] + [R^i]$.

Our Contribution: Solution to the GRP with MPC – Challenges in SODO, SODS & SSDS

1) Include all queued payments in the solution.
2) Calculate balances for all the banks,
   - If there is at least one negative balances then execute step 3.
   - If all the balances are positive then stop.
3) Choose the bank with the negative balance and remove from the last payment in queue for this bank from the solution. Repeat step 2.

- First, define an array [h] with the size of the number of parties.
- Then, calculate the balances. If the balance of bank i is positive set h_i to be 0. Otherwise set h_i to be 1.
  - $[h_i] = [B_U^i] < 0$ There are special MPC protocols to do this comparison.
- Then to determine if we have a negative balance among the banks, we compute
  - $[z] = \Pi (1- [h_i])$
  - Open [z] to reveal it
  - If z = 1, all the balances are positive and we already solved the problem.
  - If z= 0, it means that there is at least one negative balance and we should goto step 3.

Our Contribution: Solution to the GRP with MPC – Challenges in SODO, SODS & SSDS

1) Include all queued payments in the solution.
2) Calculate balances for all the banks,
   - If there is at least one negative balances then execute step 3.
   - If all the balances are positive then stop.
3) Choose the bank with the negative balance and remove from the last payment in queue for this bank from the solution. Repeat step 2.

How to remove the last transaction of the bank with the negative balance without leaking any information?

- We need to go over all transactions:
  - For v which is the number of transactions that are included in the queue do:

    For i = 1, …, v-1

    $$[x_i] = ([x_i] * [x_{i+1}]) * [h_i] + [x_i] * (1 - [h_i])$$

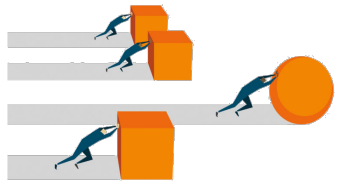
    $$[x_v] = [x_v] * (1 - [h_v])$$

  - This will set to 0 only the x_i for banks i for which the current balance is negative, and therefore, their last transaction is removed from the queue.

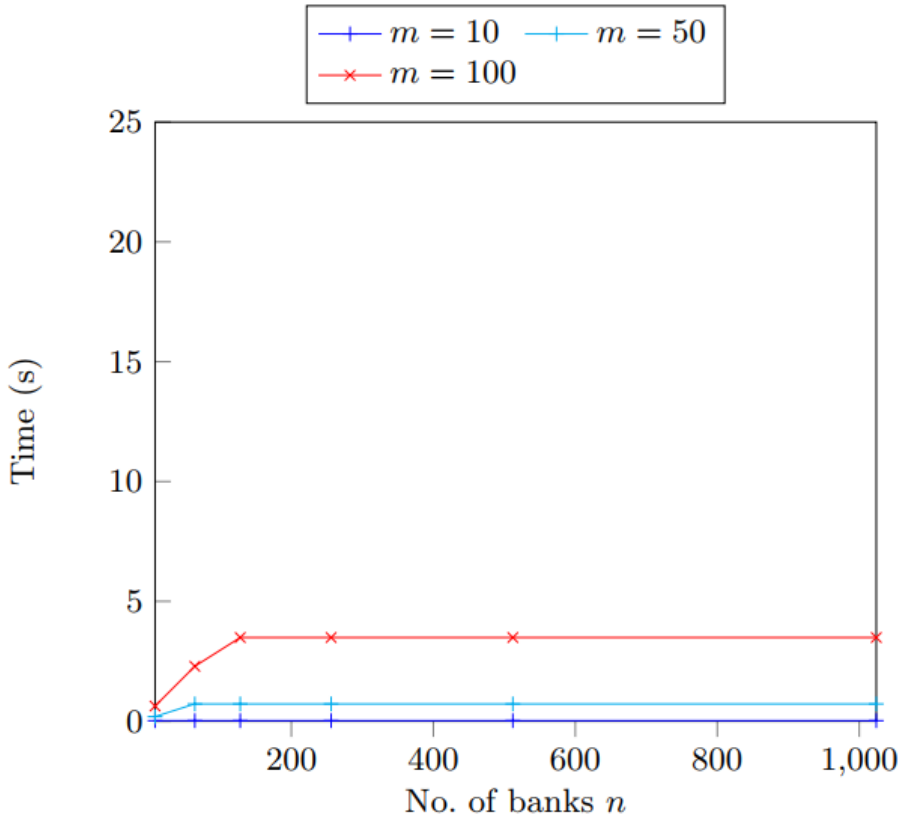
Outline:

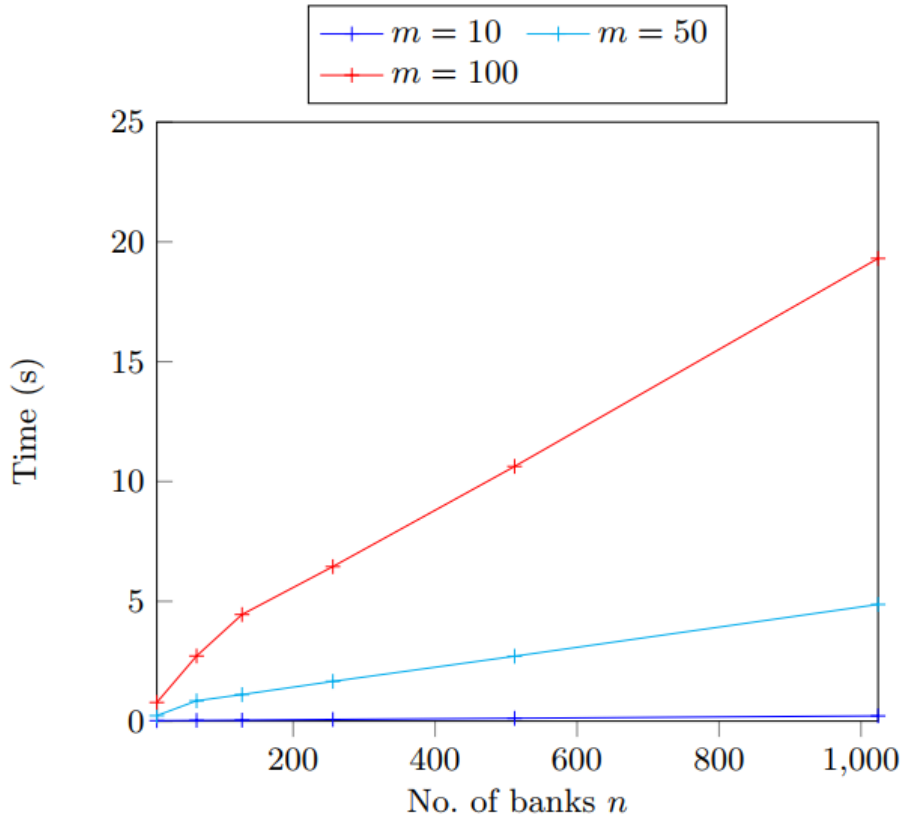
- Introduction and Background
  - RTGS systems
  - Liquidity of banks
  - GridLock Resolution Problem (GRP)
- Problem statement
  - Decentralized RTGS system
- Our Contribution
  - Main idea
  - Algorithms
    - Sources Open & Destinations Open (SODO)
    - Sources Open & Destinations Secret (SODS)
    - Sources Secret & Destinations Secret (SSDS)
  - Performance
    - Runtimes of three different algorithms
    - Simulation result

MPC in ECB Systems: Performance- SODO & SODS

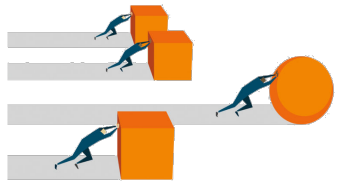


Runtimes of SODO and SODS, where n is the number of banks and m is the number of transactions to be processed. The runtimes are in second.

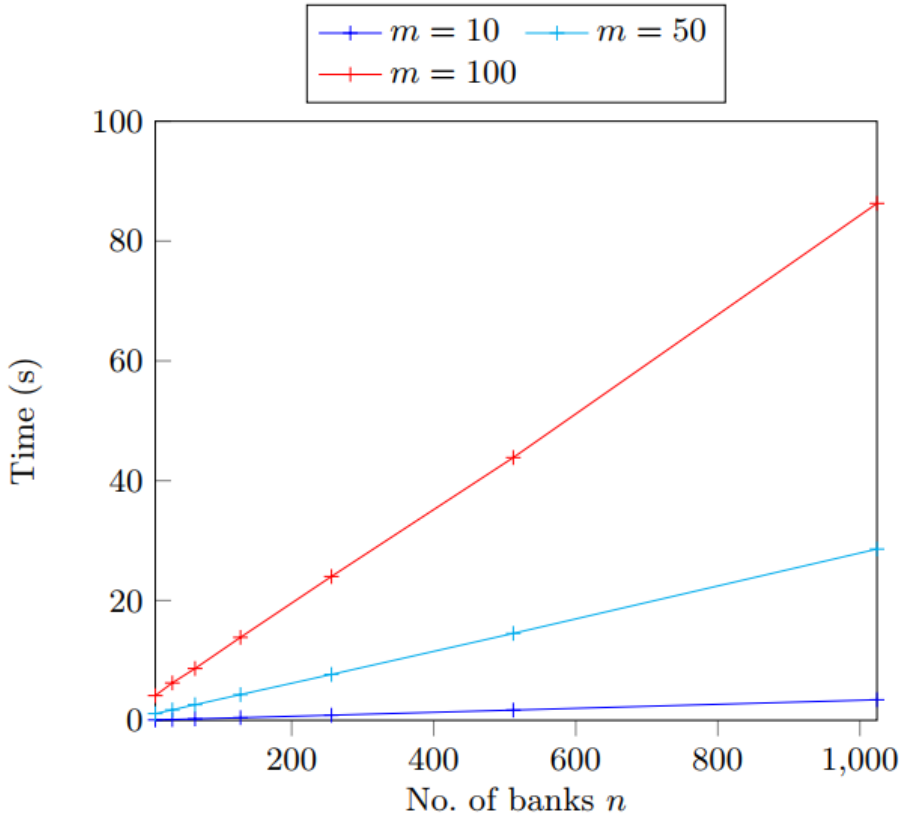




MPC in ECB Systems: Performance- SSDS



Runtimes of SSDS, where n is the number of banks and m is the number of transaction to be processed. The runtimes are in second.



MPC in ECB Systems: Performance- Simulation

The previous performance results give one times for ONE execution of the algorithm.

But in practice we care about clearing the results over a day of execution.

The value m will vary during the day.

- Depending on the transactions sizes, amounts and liquidity in the system.

So to get real run times we need to simulate a day's execution and see if the throughput can cope with the number of transactions and the number of items m in the queue.

To perform the simulation we use a simulation methodology given by Soramäki and Cook in 2013.

MPC in ECB Systems: Performance- Simulation

Methodology of Soramäki and Cook in 2013:

- Transaction graphs follow a scale free distribution
- We want to sample (s,a,r) values for the transactions where:
  - s = name of source bank
  - a = amount
  - r = name of receiving bank
- The network contains an initial set of n_0 banks that are supposed to send and receive transactions more than the other banks
  - Each bank b has a preferential attachment v_b.
  - To start the preferential attachment v_b for the n_0 initial banks is 1, and for the remaining banks it is 0.
  - Whenever q transactions are generated (q is a parameter to define), we switch the preferential attachment to 1 for one of the banks that are still not yet included.
  - Whenever a bank sends/receives a transaction, its preferential attachment grows.

MPC in ECB Systems: Performance- Simulation

The algorithm proceeds as follows:

- For $k = n_0 + 1, \ldots, n$
  - For $l = 1, \ldots, q$
    - The source bank $s \in \{1, \ldots, n\}$ is selected with the probability for $j \in \{1, \ldots, n\}$, $v_s / \Sigma v_j$.
    - The destination bank $r \in \{1, \ldots, n\}$ is selected with the probability for $j \in \{1, \ldots, n\}$, $v_r / \Sigma v_j$.
    - If we obtain $s = r$ then a new value of r is sampled in the same way, until $s \neq r$.
    - Update the preferential attachment for both the source and destination, by adding $\alpha = 0.1$ to v_s and v_r.
    - The amount a is sampled by taking a value x from the normal distribution with mean 1 and standard deviation 0.2, and then setting $a = d*\exp(x)$. Where d is the minimum of the in-out degrees of the source and destination nodes s and r.
      - Thus bigger banks make bigger transactions amounts.
  - Set the preferential attachment to be 1 for one of the banks for which v_b is still equal to 0

MPC in ECB Systems: Performance- Simulation

To simulate the execution we need to define how much liquidity is in the system.

This is controlled by a simulation parameter $\beta \in [0,1]$.

For the initial balances of the banks:

Calculate the lower and upper bounds of liquidity for each bank as follows:

- The lower bound L_i for bank i refers to the minimal initial balance that will allow the bank to settle all its transactions at the end of the time window.
- The upper bound U_i refers to an initial balance that will allow the bank to settle immediately all its transactions without having to be placed in the queue U for the gridlock execution.

Finally, the initial balance of bank i we set equal to $B^i = \beta * (U_i - L_i)$.

MPC in ECB Systems: Performance- Simulation

First generate transactions using the distribution of the simulator.

Distribute them over one hour at uniform time intervals.

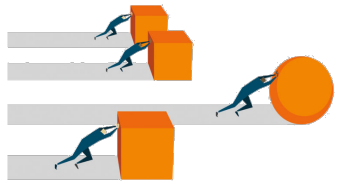
Clearing them using our algorithms using 2 versions:

1. In the first version we take the transactions one by one.
2. In the second version whenever we take transactions, we enter all the ones that arrived whilst we were executing the previous GRP step.

At the end of the processing of the hour we calculate:

- E: the Excess which is the time it took us to clear all transactions minus one hour.
  - E = 0 is perfect. The MPC variant results in no delay.
- D: the Delay which is the average delay in terms of executed time vs entered time for each transaction.
  - D = 0 is perfect. There is no delay for any transaction.

MPC in ECB Systems: Performance- Simulation



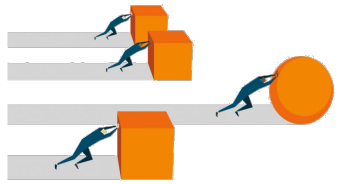
In the SODO and SODS cases, transactions could be cleared in effectively real time, with no delay due to the secure nature of the processing.

			Version 1		Version 2	
			E	D	E	D
sodoGR	$\beta = 0.1$	$n = 100,\ M = 900$	0	0	0	0
		$n = 100,\ M = 9000$	0	106	0	0
		$n = 100,\ M = 45000$	-	-	0	0
		$n = 1000,\ M = 9900$	-	-	0	1
	$\beta = 0.5$	$n = 100,\ M = 900$	0	0	0	0
		$n = 100,\ M = 900$	0	0	0	0
		$n = 100,\ M = 45000$	-	-	0	0
		$n = 1000,\ M = 9900$	-	-	0	0
	$\beta = 0.9$	$n = 100,\ M = 900$	0	0	0	0
		$n = 100,\ M = 900$	0	0	0	0
		$n = 100,\ M = 45000$	-	-	0	0
		$n = 1000,\ M = 9900$	-	-	0	0

			Version 1		Version 2	
			E	D	E	D
sodsGR	$\beta = 0.1$	$n = 100,\ M = 900$	0	1	0	0
		$n = 100,\ M = 9000$	17382	11357	0	0
		$n = 100,\ M = 45000$	-	-	0	1
		$n = 1000,\ M = 9900$	-	-	85	47
	$\beta = 0.5$	$n = 100,\ M = 900$	0	0	0	0
		$n = 100,\ M = 900$	302	346	0	0
		$n = 100,\ M = 45000$	-	-	0	1
		$n = 1000,\ M = 9900$	-	-	47	41
	$\beta = 0.9$	$n = 100,\ M = 900$	0	0	0	0
		$n = 100,\ M = 900$	0	0	0	0
		$n = 100,\ M = 45000$	-	-	0	0
		$n = 1000,\ M = 9900$	-	-	0	0

- Runtimes in seconds corresponding to 1hour of an RTGS, where the transactions are coming from simulation. n shows the number of banks, M shows the total number of transactions, and a value β controlling the amount of liquidity in the system. E and D given to 0 decimal places accuracy.

MPC in ECB Systems: Performance- Simulation



In the case of SSDS, we find a significant delay being introduced, which depends on the number of banks, the number of transactions per hour, and the overall liquidity within the system.

			Version 1		Version 2	
			E	D	E	D
ssdsGR	$\beta = 0.1$	$n = 100,\ M = 900$	49	102	0	0
		$n = 100,\ M = 9000$	149559	90404	0	17
		$n = 100,\ M = 45000$	-	-	707	490
		$n = 1000,\ M = 9900$	-	-	5507	3874
	$\beta = 0.5$	$n = 100,\ M = 900$	0	0	0	0
		$n = 100,\ M = 900$	26147	13708	0	2
		$n = 100,\ M = 45000$	-	-	319	212
		$n = 1000,\ M = 9900$	-	-	10500	4983
	$\beta = 0.9$	$n = 100,\ M = 900$	0	0	0	0
		$n = 100,\ M = 900$	0	0	0	0
		$n = 100,\ M = 45000$	-	-	0	0
		$n = 1000,\ M = 9900$	-	-	3965	1877

Conclusion

- MPC can be used to emulate the party managing the RTGS system.
  - The calculation for the GRP algorithm to do the multilateral netting can be distributed.

- The performance of the RTGS system is penalized due to the nature of MPC protocols, E.g.
  - For conditional branching, we need to evaluate both branches.
  - Memory accesses should not leak information, which results in a drop in performance.

- However, using MPC to distribute RTGS systems is still viable:
  - For the cases of SODO and SODS.
  - For SSDS there is a significant delay in clearing the transactions.

KU LEUVEN

Thank You!

nigel.smart@kuleuven.be