Compute Globally, Act Locally: Protecting Federated Systems from Systemic Threats

Arjun Narayan Antonis Papadimitriou Andreas Haeberlen
University of Pennsylvania

Abstract

Building dependable federated systems is often complicated by privacy concerns: if the domains are not willing to share information with each other, a global or 'systemic' threat may not be detected until it is too late. In this paper, we study this problem using a somewhat unusual example: the financial crisis of 2008. Based on results from the economics literature, we argue that a) the impending crisis could have been predicted by performing a specific distributed computation on the financial information of each bank, but that b) existing tools, such as secure multiparty computation, do not offer enough privacy to make participation safe from the banks' perspective. We then sketch the design of a system that can perform this (and possibly other) computation at scale with strong privacy guarantees. Results from an early prototype suggest that the computation and communication costs are reasonable.

1 Introduction

In distributed systems, federation is often an obstacle to dependability. When different parts of a system belong to different administrative domains, there is no single entity that has a view of the entire system, and domains are often reluctant to share information with each other because they can have conflicting interests. As a consequence, when a federated system faces a "global" problem that no single domain can detect by itself, the problem may not be recognized until it is too late.

Several instances of this problem have appeared in "technical" distributed systems. For instance, botnets can spread their command-and-control infrastructure across multiple networks to avoid raising suspicion [14, 17], administrative systems can drop or degrade certain types of traffic while feeling reasonably safe that the behavior will not be attributed to them [19, 5], and interdomain routing protocols can fail to find good solutions, even when they would benefit everyone involved [15]. In many cases, specialized solutions have been developed that enable the domains to coordinate on a particular instance.

But the problem is more general than this: it also affects distributed systems in the offline world, where the domains are organizations or businesses, rather than computers. A recent example is the 2008 financial crisis, in which a number of financial institutions had to be bailed out at great cost to governments around the world. At a very high level, this crisis arose because some banks were selling each other a form of insurance against problematic events (such as a price drop in the housing market), but were unable to tell whether their counterparties would actually be able to pay up if the event did occur. This information asymmetry created a risk of cascading failures: if an actual price drop forced a single bank into bankruptcy, that bank might not be able to meet its obligations, creating a liquidity problem for other banks. This might then trigger additional bankruptcies, potentially leading to a system-wide failure.

The classical approach to resolving such problems is to collect all the information at a trusted third party, which then performs some computation to look for problems, and publishes the result. This could have prevented the financial crisis, since the risk was knowable in principle: someone with access to the financial information of *all* the banks could have foreseen it [1]. However, in practice, this information is extremely sensitive because it reveals details about the banks' strategies; hence, it is unlikely that the banks would be willing to share it willingly, and a government mandate would probably be met with fierce resistance [10]. An alternative could be the use of multiparty computation (MPC) [20], and this has in fact been considered for this scenario [1]. However, naïve MPC does not scale well to large number of players (such as the global banking system) or to complex computations – and, more importantly, even MPC would not necessarily be acceptable to the banks: even if the computation itself were confidential, its *output* (perhaps a list of banks that are in danger of failing) might still allow conclusions about the sensitive inputs [10, §4.2].

In this paper, we describe a new approach to this problem that we are currently working on. Our approach is based on two key insights. First, *stability criteria can sometimes be distributed into a Pregel-style[16] computation on graphs* – in other words, we can break it down into multiple rounds and, in each round, have each participant exchange information only with some local neighborhood. This is potentially a lot more efficient than doing all-to-all MPC; it may not be possible for all scenarios, but, as we will show in this paper, it is possible in at least the banking scenario. Our second key insight is that

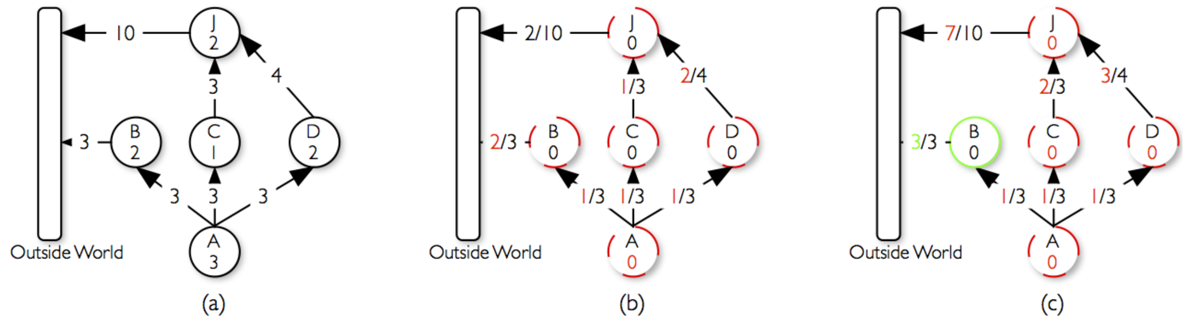

Figure 1: A simplified example over three time steps. Circles represent banks and are labeled with the liquid cash reserves; arrows represent contracts and are labeled with the agreed payout. In (b)+(c), we additionally show payments in green, and partial payouts in red. (a) shows the initial scenario, (b) the same scenario after one round of payouts, and (c) the final state after all contracts have been paid out to completion.

signs of trouble are typically large: for instance, in the banking scenario, we are not looking for small financial difficulties in individual banks, but rather for a massive systemic failure. Thus, the results do not need to be particularly precise, as long as they would reliably indicate a large problem. This imprecision greatly helps with delivering robust privacy guarantees.

In the rest of this paper, we sketch a distributed system that can support large-scale, cross-domain computations with strong, provable privacy guarantees. Our system is based on a combination of MPC, a graph-computation runtime, and differential privacy [6]. We use the 2008 financial crisis as a case study to show how our system could be applied: we adapt a computation from Bisias et al. [3, §B3] that measures the systemic risk in a network of contracts, and we demonstrate that it could be evaluated on our system within minutes on commodity hardware, using only a few MB of traffic per participant. Although our system still has some important limitations – such as limited resilience against large-scale collusion – we believe that it can be a promising step towards improving the resilience of technical and non-technical federated systems against large-scale, systemic shocks, which would result in better dependability.

2 Case study: Systemic risk in banking

We begin by describing a somewhat unusual example, the market for derivatives between banks, where dependability is an explicitly distributed criterion that cannot be broken down into locally detected conditions.

2.1 Background on financial markets

To provide some context for the following discussion, we begin by explaining a (drastically simplified) model of financial markets. Banks, as part of their regular business with clients, are exposed to *risk*. We can think of this risk as a certain event – such as a drop in house prices – that causes the bank to lose a certain amount of money. To prevent a buildup of excess exposure to any single event, banks can create *derivatives* and sell them to other parties, perhaps another bank. We can think of derivatives as "insurance contracts" that specify that a certain sum will be due if and when a particular event occurs.

A bank's exposure to a single event is often more than an order of magnitude bigger than its cash reserves; this multiple is known as *leverage*. Thus, banks depend on the derivatives to remain solvent. However, there is a risk that, when an event occurs and the debt becomes due, the other party will be bankrupt and unable to pay. This risk is known as *counterparty risk*. This risk exists even for honest banks because in practice, the chances that everything will become due at the same time are usually small; therefore, no bank keeps enough money on hand to cover *all* of its possible obligations.

Banks regularly *reinsure* their risk by buying derivatives from other banks. We can think of the resulting network of obligations as a graph in which the vertices represent the banks and the edges represent the contracts. Figure 1(a) shows a simple example of such a graph; the edges are labeled with the sum that becomes due when the event occurs, and the banks are labeled with their liquid cash reserves. This example is loosely inspired by the Credit Default Swap market from the 2008 financial crisis, with bank A representing AIG [13].

2.2 What is systemic risk?

A key challenge for banks is to be resilient to the possibility of a contract not paying out because the seller is bankrupt. While a bank might try to protect itself by diversifying its insurance purchases among a diverse group of sellers, it is hard for a bank to know who its dependencies depend upon. The reason is that each bank guards its "book" of investments closely, since this is its profit-making edge. Therefore, it is possible for almost every bank to individually believe themselves to be sufficiently diversified, but in reality they could all depend on a single upstream bank, so in effect, the diversity is zero.

The bankruptcy of a single bank is often acceptable;

the danger that regulators wish to prevent is that a single bankruptcy becomes a financial contagion that spreads to other banks, potentially causing cascading failures, and eventually financial collapse. We illustrate this problem in Figures 1(b) and (c). Figure 1(b) shows the state of the system after the event has occurred and a single round of contract payouts have been made. Notice how some banks did not have enough liquid cash to pay out all their obligations immediately (this is shown in red) – we assume that in this case payouts are made in proportion to the original amount of each contract, which is a modeling assumption supported by [3]. Figure 1(c) shows the final state once no more payouts can be made; notice how banks A, C, D, and J are unable to pay out their full obligations and thus are insolvent (shown in red at the termination of the algorithm).

We make two observations about this scenario. First, we note that a small initial problem at bank A has caused massive damage to the system; banks C and D are now also insolvent. Second, *it is very difficult to predict this based only on the local information available at each bank*. In retrospect, bank A did oversell insurance, but it is not illegal for banks to risk their own bankruptcy, and perhaps the bank's individual risk model had (incorrectly) considered the event extremely unlikely. More importantly, however, banks C, D, and J did nothing obviously wrong; they would have been fine if all their contracted payments had actually materialized. This uncertainty about the creditworthiness of other banks is thought to have contributed materially to extending the 2008 financial crisis [1].

2.3 Can the risk be detected early?

Although no individual bank can estimate the risk of cascading failures based on its local information, a hypothetical entity that has access to the combined information of *all* banks could certainly do so. Designing indices of systemic risk is a hot topic of research since the 2008 financial crisis [3], although many proposals are constrained by assumptions about the non-viability of centralization. We ignore this issue for now, but return to it in Section 2.4.

There are already some preliminary theoretical economic models [9, 2], [3, §B.3], and they share a basic structure we will now sketch briefly. The model resembles the model we have used in Figure 1: banks are modeled as nodes $n_1 \dots n_i$ in a graph, with respective starting capital $C_1 \dots C_i$, and directed edges between banks represent insurance contracts. If there are m insurable events, we have a matrix E of initial exposures, where $e_{i,k}$ represents bank i's initial exposure to event k. Banks write contracts predicated on events: a contract $d_{i,j,k,l}$ says that if event k happens, bank i pays bank j l dollars.

To estimate the contagion effects of a given subset of the events, a fixpoint iteration can then be performed on this model. In each round, banks either pay out their obligations in full, or, if they are unable to do so, reduce their payment to the actual money available. This computation eventually converges, and in the final state, the number of bankrupt banks (or, alternatively, the monetary shortfall) can then be measured. The details vary somewhat between the models; for instance, the precise payment rules used in [9] can produce multiple equilibria depending on where the computation is started, whereas the scenario in [3, §B.3] always converges to a single unique payment equilibrium. But overall, each of these models can perform fine-grained stress tests by simulating various "what-if" events (or combinations of events). Unlike the stress tests that are mandated by law today, such stress tests can take into account the precise linkages between firms.

We emphasize that these models are works in progress: they are simple, and economists are still debating the details, e.g., the exact rule sets to use. However, it seems clear that the economic theory side of the problem is in the process of being solved – in other words, we should soon know what a hypothetical centralized entity *would* compute in order to estimate systemic risk.

2.4 Strawman solutions

One obvious way to implement such a centralized entity would be to create an all-powerful government regulator. However, this does not seem practical because banks critically rely on secrecy to protect their business practices [1]. Currently, regulatory bodies that deal with *less* influential information about *single* institutions already have extremely heavy levels of legal safeguards and multiple levels of oversight [10, §3.1]. Congress is aware of the risks posed by having an all-powerful banking regulator, and thus is unlikely to create such an authority.

Another potential approach would be to use multiparty computation (MPC) [20]: one could design a circuit that has each bank's books as inputs, execute the simulation we have described above, and finally output some measure of risk, perhaps a list of banks that would fail if a certain event occurs. This approach is potentially more palatable for the banks, since they would not need to reveal their secret inputs, but it creates two additional challenges. First, MPC *does not scale well* with the number of participants, so even an implementation for a moderate number of big banks would probably be very expensive. Second (and perhaps more importantly for the banks), it *cannot guarantee confidentiality* because the output of the computation can still reveal facts about the confidential inputs through auxiliary information attacks such as in [18]. It is these two challenges that our work is intended to address.

```
# incremental computation at node i:
C'[i] = C[i] + in[0][i] + ... + in[N][i]
totalDebt[i] = debt[i][0] + ... + debt[i][N]
if(C'[i] >= totalDebt[i]) then
  F[i] = false
  for all j: out[i][j] = debt[i][j]
  for all j: debt[i][j] = 0
  C'[i] = C'[i] - totalDebt[i]
else
  F[i] = true
  prorate[i] = C'[i] / totalDebt[i]
  for all j: out[i][j] = debt[i][j] * prorate[i]
  for all j: debt[i][j] = debt[i][j] - out[i][j]
  C'[i] = 0
# final computation across all nodes:
shortfall = 0
for all i with F[i]=true:
  shortfall += debt[i][0] + ... + debt[i][N]
```

Figure 2: Pseudocode algorithm

3 Our approach

Our approach is based on two key insights. First, although MPC *generally* does not scale well with the number of participants, the algorithm from Section 2.3 is a *graph algorithm*: each round can be broken down into computations at each vertex (the banks' receipts and updates to debts and capital) and communication over the edges (the payments). Figure 2 shows the algorithm in this model; we have added a final "aggregation stage" that computes the total shortfall as a metric of systemic risk. Since we expect the vertices in the banking graph to have a relatively low degree, we can hope to run the computation by performing many smaller "blocks" of MPC in parallel, which would improve scalability.

Our second key insight is that we can potentially use *differential privacy* [6] to address the privacy concerns about the output. Differential privacy provides strong, provable privacy guarantees, which should be reassuring to the banks. Its main cost is the addition of a small amount of imprecision to the output, but, since we are looking for early warnings of large problems, a bit of imprecision (say, a shortfall of \$1 billion is reported as \$0.95 billion) should not affect the utility of the results. If a potential problem is detected, a more detailed investigation could be conducted outside of our system.

In summary, the capability we aim to provide is a privacy-enabled graph computation engine for federated systems. We note that this computation model is quite general, so our solution potentially has applications beyond risk estimation in financial networks, including the examples in the introduction.

4 Solution sketch

Overview: At a high level, our solution works as follows: Each vertex v_i in the graph is associated with a specific node i that knows its initial state; in the banking example, this might be a machine node that is controlled by the corresponding bank. However, instead of letting the nodes maintain their own state and communicate directly with each other , we replace each node i with a *computation block* B_i that consists of m different nodes, including i, and we use secret sharing [12] to split i's state among the members of B_i. This is necessary to ensure privacy: if i were allowed to see the intermediate state or the messages of "its" vertex v_i during the computation, it could learn information about other vertices.

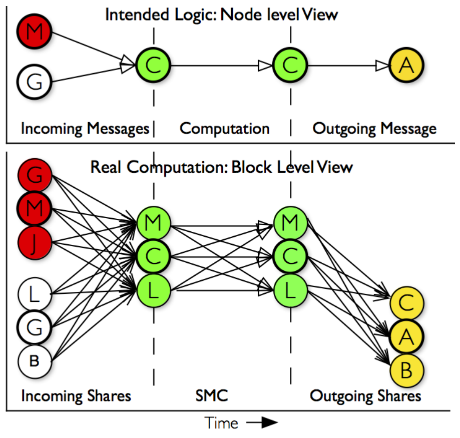

Figure 3: Example execution strategy. The top graph shows the logical computation, while the bottom graph shows the node-level computation used to execute it.

In each round, the members of B_i use MPC to compute the state updates of v_i; if v_i needs to send a message to another node v_j, the members of B_i break the message into shares and send the shares to B_j. Thus, no single node can learn anything about the state of the computation while it is still in progress; thanks to secret sharing, even $(m-1)$ colluding nodes cannot decrypt any state or any messages. Once the distributed phase of the algorithm (top part of Figure 2) has completed, each computation block sends the shares of its state to a designated computation block, which can then aggregate these states, compute the final result (bottom part of Figure 2), and finally add some imprecision as required by differential privacy, e.g., using the Laplace mechanism [8].

Figure 3 illustrates this strategy. Logically, the computation is very much like a Pregel program: vertices receive messages, update their local state, and then optionally send messages to adjacent vertices. The actual communication pattern is more complex because computations are performed by computation blocks using MPC, and messages are sent as shares.

Threat model: We assume that the players are honest but curious (HbC), although they might collude with each other. This seems appropriate for our motivating scenario. However, we speculate that malicious banks could be handled as well: MPC protocols for the fully malicious setting do exist, and if a bank reports a false initial state, this could be detected by current regulators, who already have siloed views into each institution.

Block assignment: At first glance, our solution would seem to require $n \cdot m$ nodes, where n is the number of banks and m the number of shares – but we can simply assign each node to m different computation blocks, so n nodes are sufficient. To preserve privacy, we can pick the members of each B_i from different administrative domains, so that they are unlikely to collude.

Message passing: A naïve approach for passing messages between two computation blocks B_i and B_j would be to have each member of B_i send its share to a member of B_j, who inputs it into the MPC for the next round. But this would decrease collusion resistance: m members from $B_i \bigcup B_j$ would be sufficient to decrypt the message. Instead, each node in B_i re-shares its own share to get m new shares and then sends each of them to one member of B_j. Having m^2 shares for each message may sound expensive, but in fact, with an XOR-based secret sharing scheme, the operations are quite cheap – although of course the message complexity remains at m^2.

Noising: To prevent individual nodes from controlling (or knowing) the noise that is added at the end to achieve differential privacy, we can ask each node to generate some random bits, and XOR them together in the final step to generate the random draw from the Laplace distribution. The draw itself can be performed in MPC without having to resort to expensive floating-point computations, using algorithms such as those in [7].

4.1 Initial results

To see whether our approach is practical, we built a simple prototype. Our prototype runs all the nodes on a single physical machine (an Ubuntu 12.04 Linux workstation with an Intel Core 2 Duo CPU) and performs all the computation steps sequentially – but this is sufficient for us to measure the computation and bandwidth costs. We have written custom programs to construct the boolean circuits corresponding to the systemic risk algorithms suggested by [3, §B.3] (shown in Figure 2), and the noising algorithms from [7]; for secure multi-party computation, our system uses the toolkit from Choi et al [4].

End-to-end experiment: For a small scale end-to-end run of the proposed protocols, we set up a small banking network with 10 financial institutions, using three-note computation blocks and synthetic initial data for each bank's capital and contracts. We ran 10 computation rounds, followed by the final aggregation/noising round, and we measured the amount of computation and traffic per bank. We found that each bank performed between 37.9 and 40.6 seconds worth of computation, and sent between 9.1 and 9.5 MB of traffic.

Scalability: To better gauge the computation and bandwidth costs that would be incurred in a real-size banking network, we performed a series of microbenchmarks. We constructed the circuit of the main computation step for networks of $10-100$ banks, and we varied the size of the computation blocks from 3 to 16 nodes. As expected for MPC, the bandwidth cost increased quadratically with the size of the computation blocks. In the most expensive case, a single main computation step for 100 banks with a block size of 16 nodes took 15 seconds to complete, and required the transfer of 18.31 MB per node.

Summary: Although the costs are nontrivial, we observe that the costs *per bank* depend only on the size of the computation block (which depends on the assumptions about collusion) and the complexity of the algorithm itself, but *not at all* on the overall size of the network – a substantial improvement over naïve MPC that is made possible by our formulation of the algorithm as a graph algorithm. This is in addition to the qualitative improvement with respect to privacy – recall that, unlike naïve MPC, our approach also protects against information leakage through the *output* of the algorithm.

5 Status and ongoing work

We are currently extending our prototype to a complete system that would also support other graph algorithms. Below, we sketch some of the remaining challenges.

Sensitivity: Differential Privacy requires a bound on the *sensitivity* of a computation to determine the amount of noise to add. Our algorithm's sensitivity depends on multiple factors, such as the maximum node degree and the maximum dollar value of each contract. [10] suggests *dollar-privacy*: a privacy guarantee that is weaker for larger banks and that makes sensitivity bounds easier to compute. It would also be useful to automatically infer the sensitivity from the algorithm, as shown, e.g., in [11].

Privacy budget: Differential privacy systems use a "privacy budget" that controls how much information is leaked, and that is depleted as queries are answered over time. We believe that this restriction can be relaxed in the financial setting, as there is already an expectation of "eventual disclosure" in financial regulation; thus, it may be safe to slowly replenish the privacy budget over time.

Structural privacy: Our algorithm as designed leaks some information about the graph that is not covered by the differential privacy guarantee. First, it leaks the degree of each node to neighboring nodes in the computation block as they send out shares to as many blocks as the owner node has outgoing links. Second, our block-group selection contains the owning node, which allows members of the block to narrow down the list of plausible members. We hope to mitigate or entirely prevent this in the final system.

Acknowledgments: We thank our shepherd Nalini Venkatasubramanian and the anonymous reviewers for their comments and suggestions. This work was supported by NSF grants CNS-1065060 and CNS-1054229, as well as DARPA contract FA8650-11-C-7189.

References

[1] ABBE, E., KHANDANI, A., AND LO, A. W. Privacy-preserving methods for sharing financial risk exposures. *American Economic Review: Papers and Proceedings Edition* (2012).

[2] ACEMOGLU, D., OZDAGLAR, A., AND TAHBAZ-SALEHI, A. Systemic risk and stability in financial networks. *American Economic Review* (forthcoming).

[3] BISIAS, D., FLOOD, M., LO, A. W., AND VALAVANIS, S. A survey of systemic risk analytics. *Office of Financial Research Working Paper Series*, 1 (2012).

[4] CHOI, S. G., HWANG, K.-W., KATZ, J., MALKIN, T., AND RUBENSTEIN, D. Secure multi-party computation of boolean circuits with applications to privacy in on-line marketplaces. In *Topics in Cryptology–CT-RSA 2012*. Springer, 2012.

[5] DISCHINGER, M., MISLOVE, A., HAEBERLEN, A., AND GUMMADI, K. P. Detecting BitTorrent blocking. In *Proceeding of the Internet Measurement Conference (IMC)* (2008).

[6] DWORK, C. Differential privacy. In *International Colloquium on Automata, Languages and Programming (ICALP)* (2006).

[7] DWORK, C., KENTHAPADI, K., MCSHERRY, F., MIRONOV, I., AND NAOR, M. Our data, ourselves: Privacy via distributed noise generation. In *Proceedings of EUROCRYPT* (2006).

[8] DWORK, C., MCSHERRY, F., NISSIM, K., AND SMITH, A. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography* (2006).

[9] ELLIOTT, M., GOLUB, B., AND JACKSON, M. O. Financial networks and contagion. *American Economic Review* (forthcoming).

[10] FLOOD, M., KATZ, J., ONG, S., AND SMITH, A. Cryptography and the economics of supervisory information: Balancing transparency and confidentiality. *Office of Financial Research Working Paper Series*, 11 (2013).

[11] GABOARDI, M., HAEBERLEN, A., HSU, J., NARAYAN, A., AND PIERCE, B. C. Linear dependent types for differential privacy. In *ACM Symposium on the Principles of Programming Languages (POPL)* (2013).

[12] GOLDREICH, O. *Foundations of Cryptography: Volume 2, Basic Applications*, vol. 2. Cambridge university press, 2009.

[13] HALDANE, A. G. Rethinking the financial network. *Speech at the Financial Student Association* (2009).

[14] JELASITY, M., AND BILICKI, V. Towards automated detection of peer-to-peer botnets: On the limits of local approaches. In *Proceedings of the Workshop on Large-Scale Exploits and Emergent Threats* (2009).

[15] MAHAJAN, R., WETHERALL, D., AND ANDERSON, T. Negotiation-based routing between neighboring ISPs. In *Proceedings of the Symposium on Networked Systems Design and Implementation (NSDI)* (2005).

[16] MALEWICZ, G., AUSTERN, M. H., BIK, A. J., DEHNERT, J. C., HORN, I., LEISER, N., AND CZAJKOWSKI, G. Pregel: a system for large-scale graph processing. In *SIGMOD* (2010).

[17] NAGARAJA, S., MITTAL, P., HONG, C.-Y., CAESAR, M., AND BORISOV, N. Botgrep: finding P2P bots with structured graph analysis. In *Proceedings of the USENIX Security Symposium* (2010).

[18] NARAYANAN, A., AND SHMATIKOV, V. Robust de-anonymization of large sparse datasets. In *Proceedings of the IEEE Symposium on Security and Privacy* (2008).

[19] TARIQ, M. B., MOTIWALA, M., FEAMSTER, N., AND AMMAR, M. Detecting general network neutrality violations with causal inference. In *Proceedings of the Conference on Emerging Networking EXperiments and Technologies (CoNEXT)* (2009).

[20] YAO, A. Protocols for secure computations (extended abstract). In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)* (1982).