Privacy-Preserving Network Analytics

Marcella Hastings, Brett Hemenway Falk, Gerry Tsoukalas*

August 23, 2020

Abstract

Using financial networks as a backdrop, we develop a new framework for privacy-preserving network analytics. Adopting the debt and equity models of Eisenberg and Noe (2001) and Elliott et al. (2014) as proof of concept, we show how aggregate-level statistics required for stress testing and stability assessment can be derived on real network data, without any individual node revealing its private information to any third party, be it other nodes in the network, or even a central agent. Our work helps bridge the gap between the theoretical models of financial networks that assume agents have full information, and the real world, where information sharing is hindered by privacy and security concerns.

1 Introduction

The 2008 financial crisis highlighted the fragility of existing financial networks and directly led to legislation, such as Dodd-Frank (2010), designed to identify and mitigate systemic risks. A core component of this legislation mandates that financial institutions, and banks in particular, meet capital requirements and engage in "stress-testing" against various hypothetical adverse scenarios. The effectiveness of such measures has understandably received considerable academic attention.[1]

Extant work commonly presumes that complete information about the network structure is available. This assumption is crucial because network-level dynamics depend on interactions between the institutions in the network, and emergent properties generally cannot be identified by examining each institution in isolation. In practice, however, complete information is *not* available to the institutions themselves. Each institution (presumably) knows its own assets, liabilities and interdependencies, but given these are commercially sensitive, they cannot be openly shared with

*Hastings: mhast@cis.upenn.edu, Hemenway Falk: fbrett@cis.upenn.edu, Tsoukalas: gtsouk@wharton.upenn.edu.

[1]See, e.g., Allen and Gale (1998, 2000), Morris (2000), Eisenberg and Noe (2001), Gai and Kapadia (2010), Blume et al. (2011), Gouriéroux et al. (2012), Elliott et al. (2014), Acemoglu et al. (2015), Jackson and Pernoud (2019).

the other—sometimes competing—institutions in the network.[2] What's more, security, liability and information leakage concerns can also distort information sharing incentives even in the presence of a "trusted" central agent.[3,4] This fragmentation of knowledge presents a serious barrier to understanding basic properties of the network and can even make it impossible for individual members of the network to accurately carry out mandated stress tests, assess their own risk, or even compute their own market value. To the best of our knowledge, this tension between data privacy and regulatory oversight has been highlighted in previous literature, but has not been directly addressed.

In this paper, we develop a new framework for privacy-preserving network analytics that resolves this tension. Leveraging cryptographic principles from the multi-party computation (MPC) literature, we show how critical network-level statistics can be computed without any individual node revealing its private information to any third party, be it other nodes in the network, or even a central agent. As proof of concept, we apply the algorithms we develop in network settings using real (and synthetic) data. Below, we describe the problem and our contributions in more detail.

Stress-testing with limited information

In practice, many stress tests are performed individually by each firm (e.g. those mandated by the Dodd-Frank Act Stress Tests DFAST 2019), but given the aforementioned informational limitations, these tests cannot adequately take account of network effects. This problem has been recently highlighted in Jackson and Pernoud (2019):

> "...running stress tests for each bank separately overlooks a significant source of systemic risk. Indeed, without detailed information on the overall network, a bank-specific stress test does not capture the fact that a decrease in a bank's direct asset holdings is also likely to depress other banks' values, and hence also depress the value of its inter-bank assets."

To exemplify, Figure 1 depicts a simple n-bank network where local stress testing reveals one failure, but in reality all but one of the institutions would fail. More specifically, assume bank 1 begins with reserve of 1 and all other banks have no reserves. The arcs between nodes represent

[2] In practice, direct network information about the network may be so hard to come by that firms may have to resort to inferring network structures from coverage in the mainstream news (Schwenkler and Zheng 2019).

[3] For example, in 2015, the City of Boston and the Boston Women's Workforce Council (BWWC) launched an initiative to identify salary inequities. While data owners were willing to entrust salary and other sensitive data to a third party, "one of the major hurdles [...] was the unwillingness of any single entity (including a major local university, originally enlisted to perform the study) to assume the liability in case of leakage or loss of data entrusted to them." (Bestavros et al. 2017).

[4] Another reason institutions may be reluctant to share their data with a trusted party, is that it may still be at risk of subpoena (Economist 2014).

liabilities between each bank. If bank 1's reserves drop by 10%, only bank 1 will fail its local stress test, whereas in reality, such a drop would cause n 1 cascading failures, and only bank n would remain solvent given it has no liabilities.

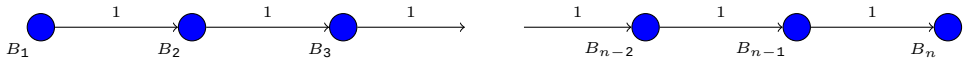


Figure 1: A n-bank example where local stress testing fails to identify systemic risk.

Local stress testing can also fail to provide information about how to structure bailouts. Figure 2 gives an example of a simple 4-bank network (a scaled version of the one described in Jackson and Pernoud (2019)) where local stress-testing gives insufficient information. Locally, banks 1 and 4 look similar in that they both have $4D$ of incoming debt, and $5D$ outgoing. Yet if both banks 1 and 4 fail (their reserves drop to 0) which bank should be bailed out? If we assume that bankrupt banks pay *none* of their debt,[5] then bailing out bank 1 by injecting D dollars saves banks 1, 2 and 3, whereas a bailout of D dollars to bank 4 saves *none* of the banks.

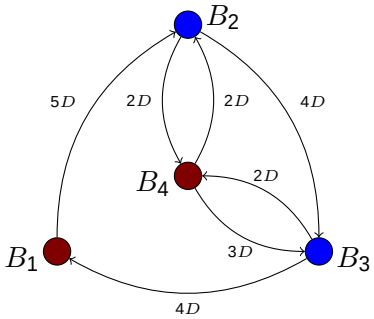


Figure 2: An example of a 4-bank network debt model adapted from Jackson and Pernoud (2019). Arrows point in the direction debt is owed.

Firm-level implications

The aforementioned limitations affect not only the effectiveness of regulations imposed by a central entity, but can also have significant adverse effects on the individual institutions themselves. Consider that, depending on the network structure, each institution may not even be able to accurately assess its own market value without knowing the assets and liabilities of *all* other institutions! What's more, certain types of networks can have high *sensitivity*, which means that small imperfections in knowledge can lead to significant errors when assessing market values (Feinstein et al.

[5]In the original debt model of Eisenberg and Noe (2001), debts were paid proportionally. Other debt models (e.g. Gai and Kapadia (2010)) have assumed a zero recovery rate as in this example.

2018).

These examples show that even in the simplest, most idealized network settings, financial institutions cannot be expected to calculate (or even accurately estimate) their own market values. To make matters worse, the problem extends to most statistics of interest to the stakeholders. Complete, perfect knowledge of the network is also necessary to assess risk, or identify business strategies. Consider simple questions like: How many institutions would default if the underlying assets experienced a uniform drop of 10% in value? Or, how much would a given firm's market value increase if an asset class increased in value by 5%? These questions cannot be answered accurately by the institutions themselves without knowing the detailed portfolios of all the institutions in the network.

In brief, the privacy and security requirements of the individual institutions in the network prevents them from engaging in cooperative risk assessment and can be a serious practical barrier to institutions and central regulators alike.

Contributions

In this work, we address some of the aforementioned issues by designing and implementing privacy-preserving multiparty computation (MPC) algorithms in two seminal network settings: the liabilities model of Eisenberg and Noe (2001) (in §4) and the equities cross-holding model of Elliott et al. (2014) (in the Appendix, §A). We outline two contributions in particular: (i) The design of novel *data-oblivious* algorithms (see Definition 2) for computing market values in network settings under privacy preservation (see Algorithms 4 and 6); (ii) a complete software implementation of our protocol, together with benchmarks of its computational and communication costs in real-world environments. We expand on each of these next.

(i) *Data-oblivious algorithms:* Data-oblivious algorithms guarantee that information isn't leaked by the algorithm operations themselves. Most existing MPC protocols focus on hiding the *data* in a computation, rather than the *operations* (see Section 3.2 and Appendix B.2 for further discussion). Thus, before an algorithm can be implemented securely, its sequence of *operations* must be made data independent. Previous works have avoided this problem by focusing on simple algorithms (e.g. securely computing means and variances) where the sequence of arithmetic operations is fixed, and hence independent of the input data. In our setting, the algorithms for computing market clearing values in network settings are significantly more complex, and the algorithms developed in the extant literature (outside the context of secure computation) are *not* data-oblivious.

(ii) *Robust multiparty implementation:* The benefits of using MPC for privacy-preserving risk assessment has been discussed in recent literature (Abbe et al. 2012, Flood et al. 2013, Cai and Kou 2019). These works, however, do not specifically focus on network settings, do not provide implementations, nor do they seek to design concrete protocols with realistic network data. Closer to our setting is Narayan et al. (2014), who conduct a preliminary study showing the feasibility of using MPC to compute stress tests in financial networks. In contrast to our paper, however, they do not seek to develop a globally secure algorithm nor benchmark it on real network data.[6] Our paper's second main contribution—a robust multiparty implementation—shows that our data-oblivious algorithms are indeed practical.

The computational and communication costs of MPC protocols has typically been a barrier to adoption. The exact cost of these protocols has a complex dependency on the underlying algorithms, the software implementation as well as the computing hardware and networking infrastructure, thus the best way to assess the practicality of an MPC algorithm is to actually implement and benchmark it. To this point, we implement and benchmark our algorithms using three different open-source software packages (each with their own advantages and disadvantages) and evaluate their suitability for practical implementation: SCALE-MAMBA (Aly et al. 2019), MPyC (Schoenmakers 2019) and EMP-Toolkit (Wang et al. 2017).

Even with the existence of these open-source tools, implementing secure multiparty computation protocols remains a significant engineering challenge, and to make our work reproducible, we have released fully self-contained Docker "images" under open-source license (Merkel 2014). The Docker images contain all the source code for our implementations and provide a consistent build environment, as well as all the libraries needed to run our examples.

In the past few years, there has been a dramatic increase in the number of *applications* and *implementations* of MPC protocols, and we highlight some of the recent applications of MPC towards privacy-preserving financial analytics here. For instance, continuing the example mentioned in footnote 3, the city of Boston together with the Boston Women's Workforce Council used MPC to calculate aggregate payroll analytics across over 150 different companies in Boston, with the aim of studying gender-based discrepancies in employee compensation (Lapets et al. 2018).

Protocols have also been developed for linking sales and purchase records in Estonia in order to identify potential cases of tax fraud without revealing the underlying sales and purchase data

[6] To avoid the costs of doing a *global* secure computation, Narayan et al. (2014), breaks the network until smaller groups of nodes, and performs secure computation within the groups. This leaves the system vulnerable if a small number of institutions *within a group* collude, and it leaks information about debts between the groups. In addition, the protocol leaks information about the whether debts exist between institutions (while hiding their magnitude).

(Bogdanov et al. 2016). Sangers et al. (2019) used homomorphic encryption, a different model of secure computation, to identify fraudulent transactions in banking networks.

Our work contributes to this literature by providing a novel set of generalizable MPC algorithms for network applications and evaluating the feasibility of implementing these algorithms using real data.

More broadly, our work contributes to a recent and growing literature studying the implications of new technologies in finance (FinTech). While the bulk of this literature so far has focused on blockchain/cryptocurrency and crowdfunding applications,[7] privacy-preserving technology is being increasingly adopted in practice, as the computational methods and algorithms underlying it continue to improve. This paper, much like Abbe et al. (2012), seeks to expand the boundaries of the extant fintech literature into this novel direction.

To summarize, the methods we develop can help resolve the tension between the privacy requirements of the individual institutions and the collective value of aggregate network statistics, in virtually any type of network setting. Our work helps to bridge the gap between our extant theoretical models of financial networks—that generally assume complete information—and the real world—where information sharing is hindered by privacy concerns.

The rest of the paper is organized as follows: §2 provides a brief overview of secure multiparty computation. §3 describes the network model of Eisenberg and Noe (2001) and the obstacles that need to be overcome for privacy preservation in this (and other) network settings. §4 details the algorithms we develop to overcome these obstacles. §5 implements the algorithms using real and synthetic data. §6 concludes. Extensions, technical details and proofs are provided in the Appendix.

2 Secure Multiparty Computation (MPC)

Secure Multiparty Computation is a cryptographic technique that, in theory, allows a group of data owners to securely compute any function of their private data, without revealing their underlying data to each other or to any outside party, and without the need for a central agent. The first general-purpose cryptographic MPC protocols were developed in the 1980s (Goldreich et al. 1987, Ben-Or et al. 1988, Chaum et al. 1988), and since then, MPC has been an active area of research. Below, we go over some of the basic principles and definitions that we require. The technical details regarding how to conduct some of the privacy-preserving operations required for network settings

[7] For instance, see Babich et al. (2020), Tsoukalas et al. (2019), Belavina et al. (2020), Biais et al. (2019), Chakraborty and Swinney (2020), Chod and Lyandres (2020), Chod et al. (2020), Cong et al. (2020), Gan et al. (2020), Hinzen et al. (2019), Rosu and Saleh (2020), Tsoukalas and Falk (2020)

(e.g., matrix multiplications) are left for the Appendix, §B.1.

2.1 Security

A multiparty computation protocol is called *secure* if the execution of the protocol itself reveals nothing more about the participants' (private) inputs than what is revealed by the output alone. In protocols where there is no output (or the output is secret-shared), the participants should learn *nothing* about the inputs. Roughly, this means that each player's *view* of the protocol (*i.e.*, the messages they send and receive) should be independent of the other participants' private inputs (after conditioning on the output).

Roughly multiparty computation protocol is said to be secure if the views of the players (*i.e.*, the messages sent and received) by running the protocol on two different sets of inputs are indistinguishable (see Appendix B.4 for a formal definition of indistinguishability, and further discussion of security in multiparty protocols). To formalize this notion of security, in a setting with n partcipants, define, for $i = f1, \ldots, ng$, a function of interest that needs to be computed in a privacy-preserving way:

$$f_i : X^n \; ! \; Y$$
$$(x_1, \ldots, x_n) \; 7! \; y_i.$$

The variables x_i, y_i are the input and output of participant i (out of n). Note that in many scenarios all f_i are equal, *i.e.*, all participants receive the same output.[8]

The *view* of player i in a multiparty protocol on inputs $x_1, \ldots, x_n$, denoted $\mathsf{view}_i(\vec{x})$, is the collection of all messages sent and received by player i when the protocol is run on input $\vec{x}$. Because secure computation protocols are randomized, for any fixed $\vec{x}$, $\mathsf{view}_i(\vec{x})$ is a random variable.

A protocol is (t-n)-secure if any set of at most t participants can gain no information about the remaining $n \quad t$ players' private inputs by colluding together. When $t = n \quad 1$, we have the strongest notion of security, *i.e.*, each individual's privacy is preserved even if *all* other participants collude against her. Intuitively, we have (t-n)-security if the view of any subset of (at most) t players, when the protocol is run on one set of inputs, conditioned on their outputs, is "indistinguishable" (Definition 7) from their view when the protocol is run on a different set of inputs. This notion is formalized in Definition 1.

[8] In our settings, f_i could be the function that computes the current market value of institution, i. If all participants wanted to learn all market values, then all f_i would be the same vector-valued function that returns the vector of all market values

Definition 1 (Secure multiparty computation (indistinguishability)). *An n-party computation protocol for computing $f \stackrel{def}{=} (f_1, \ldots, f_n)$ is secure $(t\text{-}n)$-secure against passive (semi-honest) adversaries if for any $T \subseteq [n]$, with $|T| \leq t$, and any inputs $\vec{x}, \vec{x}'$ such that $f_i(\vec{x}) = f_i(\vec{x}')$ for all $i \in T$, the random variables*

$$\left\{ \text{view}_i(\vec{x}) \right\}_{i \in T} \mid \left\{ y_i \right\}_{i \in T} \quad \left\{ \left\{ \text{view}_i(\vec{x}') \right\}_{i \in T} \mid \left\{ y_i \right\}_{i \in T} \right\}, \tag{1}$$

where $\equiv$ denotes indistinguishability of distributions*, formalized in Definition 7. The protocol is* perfectly secure *if, conditioned on the outputs $\{y_i\}_{i \in T}$, the views of the players in T are independent of the inputs.*

Remark 1. *Definition 1 ensures that the protocol leaks no more information* than the output alone. *On the other hand, MPC does not provide security against information that can be gleaned from the output alone. For example, consider the special case with only two participants, seeking to securely compute their average salary. By observing the output, that is, the average salary of the two, each individual could then infer the other's salary, no matter what protocol was used for the secure computation. Protecting against inference from the* output *of a computation is a separate problem, and usually requires adding noise to the results (Dwork 2011).*

Other standard notions of security in MPC protocols are discussed in Appendix B.4.

2.2 Cryptographic Secret Sharing

Most MPC protocols rely on cryptographic secret sharing (Beimel 2011), which allows a data owner to split her data into *shares*, each of which reveal nothing about the underlying data. These shares are distributed to the participants in the protocol, and the (secure) computation proceeds on the shares, rather than the underlying data itself. The simplest method for secret-sharing a secret $s \in \mathsf{F}$ among n participants is called *additive secret sharing*. The secret $s \in \mathsf{F}$, is shared by selecting n uniformly random values $r_1, \ldots, r_n \in \mathsf{F}$ subject to the constraint that $s = \sum_{i=1}^{n} r_i$, and the share r_i is given to participant i. Each r_i is called a *share* of s, and it is easy to see that any subset of at most $n - 1$ of the r_i reveals nothing about the secret, s, whereas s can be reconstructed from the collection of all n shares. From a statistical standpoint, the marginal distributions of the r_i are independent of the secret, s, while the secret s can be recovered from any sample from the joint distribution.

Example 1 (Securely computing the average). *Suppose n institutions, each have private assets $D_i \in \mathsf{Z}$, and they wish to (securely) compute their average asset values, $\frac{\sum_i D_i}{n}$, without revealing*

their underlying assets, D_i. The players first agree on a public prime, p,[9] *with p* $\sum_i D_i$. *Then, for* $i = 1, \ldots, n$ *player i generates secret shares of* D_i *as follows. Player i generates* r_{ij} *uniformly at random from* $\mathsf{Z}/p\mathsf{Z}$ *for* $j = 1, \ldots, n$ 1 *and sets* $r_{in} = D_i \sum_{j=1}^{n} {}^{1} r_{ij} \mod p$. *Thus,* $\sum_{j=1}^{n} r_{ij} D_i \mod p$, *and* r_{ij} *is the* j*th* share *of* D_i. *Player i then sends* r_{ij} *to player j. At the end of the sharing phase, player i has* r_{ji} *for* $j = 1, \ldots, n$.

Once player i has n shares $\mathsf{f} r_{ji} \mathsf{g}_{j=1}^{n}$, *player i calculates* $t_i \overset{def}{=} \sum_{j=1}^{n} r_{ji} \mod p$. *At this point, the set* $\mathsf{f} t_i \mathsf{g}_{i=1}^{n}$ *form additive shares of the sum* $\sum_{i=1}^{n} D_i$, *because*

$$\sum_{i=1}^{n} D_i = \sum_{i=1}^{n} \sum_{j=1}^{n} r_{ij} = \sum_{j=1}^{n} \sum_{i=1}^{n} r_{ij} \mod p.$$

Finally, player i broadcasts t_i *to all players. Each player then locally computes the average as* $s \overset{def}{=} \sum_{i=1}^{n} t_i \mod p$, *and computes the average as* $\frac{s}{n}$ *(note that n is public).*

This type of calculation is particularly easy under MPC because it is linear, i.e., *it only requires secure additions, and no secure multiplications or divisions. On the other hand, securely* multiplying *shares requires communication, and the running time of secure computation protocols (in the circuit model) are dominated by the number of secure* multiplications *they perform.*

Once each data owner has secret shared his or her private data among the set of participants in the protocol, the participants can compute *on the shares* in a privacy-preserving way. Early MPC protocols (e.g. Goldreich et al. (1987), Ben-Or et al. (1988), Chaum et al. (1988)) provided methods to securely add and multiply secret-shared values (and obtain secret-shares of the result). Using these simple primitives, more complex functionalities could be securely executed by first writing the function as a boolean circuit (consisting of AND, XOR and NOT gates) or an arithmetic circuit (consisting of addition and multiplication gates over a finite field, F) and then securely executing the circuit gate by gate.

In line with these principles, in our framework, the participants begin by secret-sharing their (private) liabilities (in the debt-model) or equity-holdings (in the equity model). At the end of the protocol, each institution will hold secret-shares of the vector of market values. The institutions can then use these shares to allow a certain party (e.g. a regulator) or parties (e.g. the institutions themselves) to reconstruct either some or all of the market values. The details are provided in §4.

[9] All computations are done modulo p, instead of over Z because there is no uniform distribution over Z. In particular, if r is chosen uniformly from $\mathsf{Z}/p\mathsf{Z}$, then $r + s \mod p$ is uniformly distributed, *independent of* s. The prime p is chose large enough so that there is no wrap-around in the desired computation.

2.3 MPC Protocols

The original MPC protocols were an important theoretical breakthrough, but were too inefficient for practical applications. One challenge is that all the original protocols assumed that the function being computed was written as a circuit. Unfortunately, the circuit representation of many common functionalities can be quite complex. In fact, *the boolean circuits representing the market-value computations in our network setting would require hundreds of millions of gates* which makes developing these circuits by hand infeasible, and makes it challenging to *execute* them securely. (see Section 5).

Currently, there are several general-purpose MPC compilers in the cryptographic literature (see Hastings et al. 2019 for a survey). These compilers take code written in a high-level language (often a C-like language), convert this code into an intermediate representation (usually a circuit) then execute this intermediate representation securely using an existing MPC protocol. Most compilers (e.g. Franz et al. (2014), Zahur and Evans (2015), Wang et al. (2015), Songhori et al. (2015), Demmler et al. (2015)), however, support only two-party computation, and are thus not appropriate for network settings. A few, including EMP-toolkit, Wang et al. (2017), SCALE-MAMBA Aly et al. (2019), and MPyC Schoenmakers (2019) support computations involving more than two participants, and we thus focus on these three toolkits in our implementation examples.

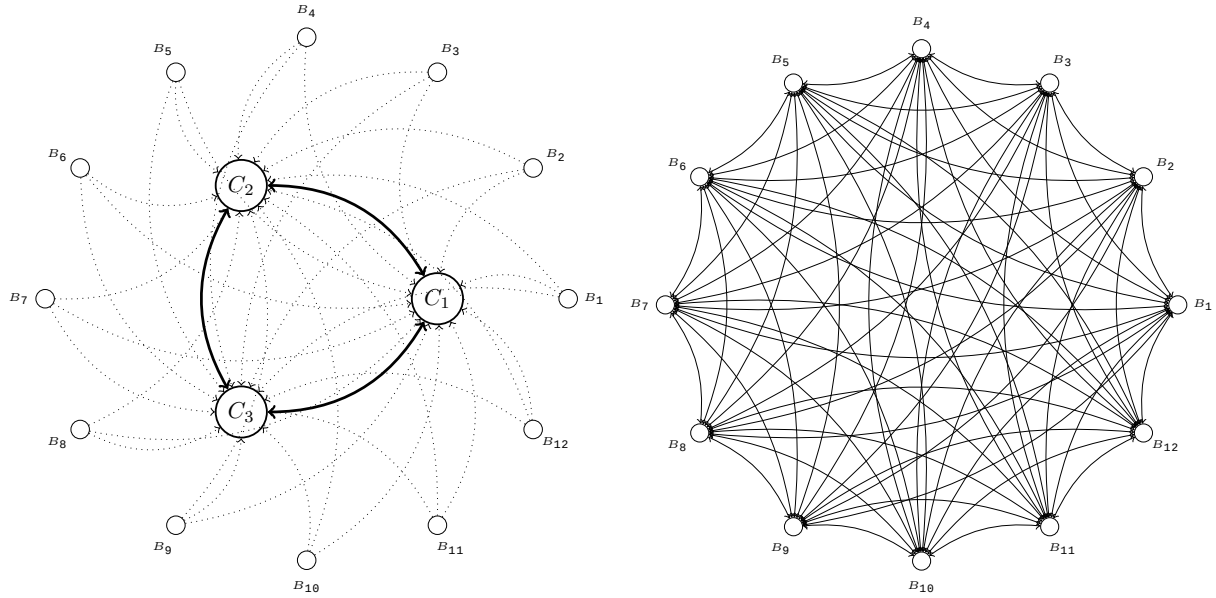
As mentioned, for security, MPC protocols can only execute programs that are *data oblivious* (see Section 3.2). This means that MPC compilers must also ensure that the programs being executed are data-oblivious. Most MPC compilers do this by severely restricting the high-level programming language. For instance compilers like EMP and MPyC do not allow conditionals on private data, instead they only provide syntax for multiplexing (see Appendix B.2.1). Similarly, none of the compilers we tested support "for" loops with *private* input bounds. Thus even with these compilers, the user is responsible for developing the *data-oblivious* algorithms that will serve as inputs (see Section 3.2).

2.4 Outsourced computation

Secure computation protocols require repeated rounds of communication between *all* of the participants. Thus, if there are n participants in the protocol, the communication complexity grows as $O\left(n^2\right)$. To avoid this, it is common to decouple the number of participants in the computation from the number of participants contributed (private) information using cryptographic secret sharing.

For example, in a network with 1000 banks, the banks could choose 3 of the larger banks to be

the "computation parties", and secret-share all private to these three privileged banks. The three banks could then engage in a secure computation to compute the vector of market values. Since the secure computation only requires three participants (instead of 1000) the overall computation could be much faster. The drawback is that this type of outsourcing changes the security guarantees, and collusion between 2-out-of-3 of the computational parties could break the security (instead of collusion between 501 of the input parties). The different architectures are outlined in Figure 3. We will utilize this method in some of our implementations in §5.



(a) Outsourcing computation to 3 participants. Dotted lines represent secret-sharing.

(b) A communication diagram of secure computation with 12 participants.

Figure 3: Outsourcing computation to a small number of parties drastically reduces the communication complexity of the protocol, but makes it more vulnerable to collusion.

3 Network Model & Obstacles to Privacy Preservation

For expositional clarity, the rest of the paper focuses on the debt model proposed by Eisenberg and Noe (2001). We treat the case of the equity model of Elliott et al. (2014) in the Appendix, §A.

Before diving into the details, we note that the underlying methods we develop could be applied to compute network statistics in essentially any network model. To this point, we provide in Appendix §A.4, an overview of several well-established financial network models that could potentially fit into our framework (albeit with some required fine-tuning).

3.1 The Eisenberg & Noe (2001) Debt Model

Consider a system with n financial institutions that have *reserves* and *debts*. Institution i's reserves are represented by e_i, and the debts (liabilities) are represented by L_{ij}, indicating a debt from institution i to institution j of L_{ij} dollars. Thus the entire network is a weighted, directed graph, where financial institutions are represented as vertices and debts are represented as (weighted, directed) edges between the institutions.

Define $\bar{p}_i$ to be the total liabilities of institution i, *i.e.*, $\bar{p}_i \stackrel{\text{def}}{=} \sum_j L_{ij}$, and the proportional liabilities

$$\Pi_{ij} = \begin{cases} \frac{L_{ij}}{\bar{p}_i} & \text{if } \bar{p}_i > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

In this proportional specification, failed or bankrupt institutions make partial payments on their outstanding debts, and the same proportion of every debt is assumed to be paid.

The total amount of money in the system is the sum of the reserves of all the stakeholders, and in this simple model, money is neither created nor destroyed. If every institution has enough reserves to pay all its debts, then each institution's market value can be computed locally by adding its reserves and incoming debt obligations and subtracting its outgoing debts. On the other hand, if some institutions fail (i.e., they do not have enough money to cover their debts) other institutions that may appear to be solvent (based on their local view) may actually be insolvent (if their incoming debts are not paid in full). This highlights the essential networked nature of the stress test – *institutions cannot effectively assess their risk based on knowledge of their debts alone.*

Formally, the action of institution i can be characterized by its total payment, p_i. We use $\vec{p}\ 2\ \mathsf{R}^n$ to denote the vector of *total* amounts paid by each institution. If $p_i < \sum_j L_{ij}$ then institution i is said to *fail* or *default*. Assuming that defaulting institutions must pay as much as they can, we have the following equations.

$$p_i = e_i + \sum_j p_j \Pi_{ji} \text{ for defaulting institutions} \tag{3}$$

$$p_i = \bar{p}_i \text{ for non-defaulting institutions.} \tag{4}$$

Computing the market-clearing vector absent privacy concerns

In this model, every institution will have a uniquely-defined equilibrium market value, based on its remaining reserves after all possible debts are settled. Assuming all information is public, Eisenberg and Noe (2001) suggest a simple, iterative algorithm for calculating the market clearing vector.

Algorithm 1 Calculating the equilibrium payment vector in a network (Eisenberg and Noe 2001)

1: Input: A network with assets $\vec{e}\ 2\ \mathsf{R}^n$, and debts $\mathbf{\Pi}\ 2\ \mathsf{R}^{n\ \ n}$
2: Initialize defaulter list, $D = ;$
3: Initialize payment vector $\vec{p} = \bar{p}$ ▷ Everyone attempts to pay full debts
4: Update the defaulter list assuming payments $\vec{p}$
5: **if** Defaulter list grows **then**
6: Update $\vec{p}$ and Go To Line 4
7: **else**
8: **return** $\vec{p}$
9: **end if**

From here, a natural network-scale stress-test can be run by adjusting (decreasing) the underlying reserves held by different institutions and calculating the number of failures (or the financial shortfall) that arise in equilibrium. This type of stress testing, however, requires knowledge of the debt structure of all institutions in the network. Performing this type of stress test using standard tools would then require all institutions to share all the details of their reserves and liabilities with some centralized regulator or testing authority who could perform the test.

What's more, this model is very sensitive to small changes in the debt structure (Feinstein et al. 2018), so high-fidelity data about the debt structure is required for effective stress testing. In particular, regulators who have only rough estimates of the inter-bank liabilities cannot effectively compute the equilibrium market values in the network.

In §3.2, we outline the specific challenges we face to develop privacy-preserving protocols in this network setting, and develop our solution in §4.

3.2 Obstacles to Privacy Preservation

In principle, *any* function can be computed securely using one of the early MPC protocols (e.g. Goldreich et al. (1987), Ben-Or et al. (1988), Chaum et al. (1988)). Unfortunately, there are many obstacles to deploying MPC in *practice*.

Practical obstacles

Efficiency: Early MPC protocols were efficient in the sense that they ran in *polynomial time*, but the actual cost (in terms of computation and communication) was too high for practical applications. Decades of algorithmic improvements, coupled with a steady increase in raw computing power

has made MPC practical for a variety of real-world problems. Nevertheless, efficiency remains a primary obstacle when developing and deploying novel MPC protocols. The practical efficiency of MPC protocols depends on a number of factors beyond the basic algorithms, including computing hardware, networking environment, software implementation, and the topology of the circuit being executed. This means that effectively measuring the efficiency of an MPC protocol requires actually implementing and benchmarking the protocol under real-world conditions.

Implementation: Despite the progress that has been made, implementing MPC protocols remains a significant engineering challenge. These are multi-participant networked protocols that require several rounds of communication interleaved with complex local cryptographic operations. Until recently, most MPC protocols were only of theoretical interest, and very few were ever implemented. As MPC protocols have improved, there has been a significant effort to implement and benchmark novel MPC protocols.

Conceptual obstacles

Designing data-oblivious algorithms: In general, MPC protocols are designed to execute algorithms that are *data-oblivious*, *i.e.*, algorithms whose control flow does not depend on their (private) inputs (Mitchell and Zimmerman 2014). Formally:

Definition 2 (Data-oblivious algorithms). *An algorithm is called* data-oblivious *if the* control-flow *(*i.e.*, the sequence of computations and memory accesses made by the algorithm) is independent of the algorithm's inputs.*

For example, if an algorithm's runtime depends on the input data, executing the algorithm leaks information about its inputs, even if the internal state of the algorithm is hidden. Developing a compact, data-oblivious representation of a given functionality or algorithm is a fundamental challenge that must be overcome before executing a computation securely.

One method for ensuring that an algorithm is data-oblivious is to represent it as a *circuit*, either a boolean circuit (consisting of AND, XOR and NOT gates), or an arithmetic circuit (consisting of addition and multiplication gates over a finite field). Most secure computation frameworks (e.g. Yao (1982, 1986), Ben-Or et al. (1988), Goldreich et al. (1987), Chaum et al. (1988)) take this approach, and these frameworks can only securely execute *circuits*.[10] Thus before a function can be computed securely, it must be represented as a *circuit* (e.g. using only boolean AND and XOR gates).

[10]There has been significant work developing secure multiparty computation protocols for RAM-model computations, but these protocols are extremely complex, and are only useful in settings which are particularly ill-suited to circuit-model computations Lu and Ostrovsky (2013), Liu et al. (2014), Boyle et al. (2015), Garg et al. (2016).

Although, in principle, any algorithm can be represented as a circuit, the circuit representation may be large, and finding the *minimum* circuit representation of an algorithm is difficult (Hirahara et al. 2018). We describe some of the common challenges and solutions in Appendix B.2.
The development of MPC compilers (Section 2.3) which can translate a high-level programming language into circuit and execute it securely represent a crucial step towards filling this void.

To this point, we outline the main challenges that exist in our network setting. The Eisenberg and Noe (2001) Algorithm 1 was not designed with privacy in mind, and thus is not data-oblivious. For example, in line 5, the algorithm terminates when the defaulter list stabilizes. Thus the number of iterations of the algorithm depends on the (private) debt-holdings. More subtly, Line 6 traditionally requires a matrix inversion (an operation useful in many network settings), and most algorithms for matrix inversion (e.g. row-reduction) are *not* data-oblivious.

Overall, there are three main issues that we need to overcome in order to make the algorithm data oblivious.

1. `Line 4`: How can the defaulter list be updated when the current payment vector $\vec{p}$ must remain private? The current payout vector depends on the debts of all the participants in the network, so updating the defaulter list requires testing whether an institution fails under a *private* payment vector $\vec{p}$.

2. `Line 5`: A data-oblivious algorithm cannot change its control flow based on private information, so when calculating the clearing vector, the termination condition cannot depend on the number set of failures.

3. `Line 6`: How can the current payout vector, $\vec{p}$, be updated obliviously? Updating the payout vector $\vec{p}$ requires solving Equation 9, which depends on the current payout vector, as well as the current set of failed institutions, both of which must remain private.

One implication of making an algorithm data-oblivious is that its running time becomes independent of the underlying data. For instance, when calculating the market values of a large network, if all the banks had zero liabilities and zero assets, a traditional algorithm might take a "shortcut" and return that all market values are zero. On the other hand, a data-oblivious algorithm *must* perform exactly the same computations when the inputs are all zero as it does on real-world inputs. One consequence of this is that all the running times we report would be the same whether we used real-world data, or simply ran the algorithms on an all-zero input. This is an instance of sacrificing efficiency for security – in the real-world, both software and hardware implement computational

"shortcuts" that can drastically improve performance when the inputs satisfy certain criteria. Unfortunately, if these shortcuts cannot be applied to *all* inputs, the fact that a shortcut was taken leaks information about the inputs, and hence cannot be applied in a privacy-preserving manner. In other words, data-oblivious algorithms can never be faster than the "worst-case" running time of its non-oblivious counterpart.

4 Data-oblivious Network Analytics

In this section, we describe the algorithms we develop for computing the defaulter list in a *data-oblivious* manner and explain how we can address the issues raised in §3.2.

These algorithms (Algorithms 2, 3, 4) do *not* provide security by themselves, instead they are only *data-oblivious*. In order to provide security, the operations required (additions, multiplications, etc.) at each step of the algorithms, need to be executed securely e.g. using the Secure Matrix Arithmetic we detail in Appendix B.1 or using the open-source MPC protocols described in §2 (as we do in the implementations).

Recall the payment equations (3) and (4) from §3. These can be written more succinctly as

$$p_i = \min\left(\bar{p}_i, e_i + \sum_j p_j \Pi_{ji}\right). \tag{5}$$

For a given payment strategy, $\vec{p}$, we can define the defaulting matrix

$$\Lambda(\vec{p}) = \begin{cases} 1 & \text{if } i = j \text{ and } i \text{ defaults under } \vec{p} \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

If complete information about the network (including all reserves and debts) were known, then the equilibrium values of each institution, the total number of failures and the total shortfall could be computed using Algorithm 1.

As noted in §3.2, the three major obstacles to making Algorithm 1 data-oblivious are: updating the current payout vector; updating the defaulter list; identifying termination conditions. Below, we outline the methodology we suggest to overcome these obstacles, culminating in Proposition 2 which shows that our solution efficiently computes an approximation to the true clearing vector and market valuations.

Payout vector For a fixed defaulting set given by Λ, the equilibrium payouts are obtained by

solving the linear equations

$$\vec{p} = \Lambda \left[\Pi^T \left(\Lambda \vec{p} + (I \quad \Lambda)\bar{p} \right) + \vec{e} \right] + (I \quad \Lambda)\bar{p}, \tag{7}$$

where Λ is the diagonal matrix with $\Lambda_{ii} = 1$ if and only if institution i has failed, Π is the $n \quad n$ proportional liability matrix, $\bar{p}$ is the debt vector, and $\vec{e}$ is the vector of underlying assets. Given a fixed defaulting set, an equilibrium payout strategy is a solution to Equation 7. Rewriting this equation, we have

$$\vec{p} = \Lambda \Pi^T \Lambda \vec{p} + \Lambda \Pi^T (I \quad \Lambda)\bar{p} + \Lambda \vec{e} + (I \quad \Lambda)\bar{p}. \tag{8}$$

Defining

$$\begin{aligned} \mathbf{A} &\stackrel{\text{def}}{=} \Lambda \Pi^T \Lambda \\ \vec{b} &\stackrel{\text{def}}{=} \Lambda \Pi^T (I \quad \Lambda)\bar{p} + \Lambda \vec{e} + (I \quad \Lambda)\bar{p}, \end{aligned}$$

Equation 8 becomes

$$\vec{p} = \mathbf{A}\vec{p} + \vec{b}. \tag{9}$$

This gives the solution

$$\vec{p} = (\mathbf{I} \quad \mathbf{A})^{\ 1} \vec{b}. \tag{10}$$

As discussed in §3.2, traditional matrix inversion algorithms (e.g. Gaussian Elimination) are not data-oblivious, and thus cannot be implemented directly as a secure computation. In Appendix §B.1.3, we review some of the special-purpose, secure matrix inversion protocols that avoid the circuit model altogether.

To address this, we use Algorithm 2, below, which outlines a simple, iterative method for (approximately) solving Equation 9. Subsequently, in Proposition 1, we show that Algorithm 2 will converge to the correct result given enough iterations or under special circumstances. We report on empirical convergence speeds in the next section. Algorithm 2 is data-oblivious in the sense of Definition 2 because the number of rounds of the "for"-loop, k, and the actions sequence of multiplications does not depend on the inputs, $\mathbf{A}, \vec{b}, \vec{p}^{\mathbf{0}}$. This overcomes obstacle 3 as the *sequence* of multiplications and additions is independent of the inputs.

Algorithm 2 FindFix

Input: $\mathbf{A}, \vec{b}, \vec{p}^{(0)}$

for $i = 1, \ldots, k$ do

$\vec{p}^{(i)} = \mathbf{A}\vec{p}^{(i-1)} + \vec{b}$

end for

return $\vec{p}^{(k)}$

Proposition 1 (FindFix Algorithm Properties). *(i) Let* $\mathbf{A} \in \mathbb{R}^{n \times n}$ *be a matrix with spectral radius* $\rho \stackrel{def}{=} \rho(\mathbf{A})$. *If* $\rho < 1$ *then* $\vec{x} = \mathbf{A}\vec{x} + \vec{b}$ *has a unique solution,* $\vec{x}$ *, and Algorithm 2 will return an approximation,* $\vec{y}$*, such that as the number of iterations,* $k \to \infty$*,* $\vec{y} \to \vec{x}$ *. In addition, if* $\vec{p}^{(0)} \geq \vec{x}$ *, then* $\vec{p}^{(k)} \geq \vec{x}$ *for all* k*. (ii) In an acyclic debt network with* n *nodes, Algorithm 2 (with* $k = n$*) solves Equation 9 with* no *error.*

Defaulter list Next, to deal with obstacle 1, we develop Algorithm 3, below, which takes a proportional liability matrix, $\mathbf{\Pi}$, an asset vector $\vec{e}$, and a proposed payout vector, $\vec{p}$, and calculates which institutions will fail under the proposed payout vector $\vec{p}$. If institution i fails, Λ_{ii} is set to 1. Looking at Algorithm 3, the number of iterations of the "for" loop, n, is public, and the calculation of the payout, $e_i + \sum_j p_j \Pi_{ji}$, is easily expressed as an arithmetic circuit. Thus Algorithm 3 becomes entirely data-oblivious if the conditional can be evaluated obliviously. In practice, this is easily achieved by evaluating both branches of the conditional at each step and *multiplexing* the result. See Appendix B.2.1.

Algorithm 3 UpdateLambda

Input: $\mathbf{\Pi}, \vec{e}, \vec{p}$

for $i = 1, \ldots, n$ do

if $p_i > e_i + \sum_j p_j \Pi_{ji}$ then

$\Lambda_{ii} = 1$

else

$\Lambda_{ii} = 0$

end if

end for

return Λ

Market values (main algorithm) Putting these pieces together, we propose Algorithm 4 to

compute the market values of the institutions in the debt network model *obliviously*. To overcome the early termination problem (Obstacle 2) we run the algorithm for n steps, independent of the inputs, and show in Proposition 2 that after n steps the defaulter list will *always* have stabilized.

Algorithm 4 A data-oblivious algorithm for calculating the market values in a debt network.

1: Input: $\vec{e}$ 2 R^n, $\mathbf{\Pi}$ 2 R$^{n \quad n}$
2: Initialize $\Lambda = \mathbf{0}$ 2 R$^{n \quad n}$ ▷ No defaults yet
3: Initialize $\vec{p} = \bar{p}$ ▷ Everyone attempts to pay full debts
4: for $i = 1, \ldots, n$ do
5: $\mathbf{A} = \Lambda \Pi^T \Lambda$
6: $\vec{b} = \Lambda \Pi^T (I \quad \Lambda)\bar{p} + \Lambda \vec{e} + (I \quad \Lambda)\bar{p}$
7: $\vec{p} =$ FindFix$(\mathbf{A}, \vec{b}, \vec{p})$ ▷ Find equilibrium with this defaulter list (calls to Algorithm 2)
8: $\Lambda =$ UpdateLambda$(\mathbf{\Pi}, \vec{e}, \vec{p})$ ▷ Update defaulter list (calls to Algorithm 3)
9: end for
10:
11: for $i = 1, \ldots, n$ do
12: $v_i = e_i + \sum_j p_j \Pi_{ji} \quad \bar{p}_i$ ▷ Market value is assets plus payments minus liabilities
13: end for
14: return $\vec{v}$

Remark 2. *For financial stress testing, it may be desirable to calculate the number of defaulting institutions or the total shortfall of the system, rather than the market values of the institutions. Since the number of shortfalls is simply* $\mathsf{j}\,\mathsf{f} v_i \;\mathsf{j}\; v_i < 0 \mathsf{g} \mathsf{j}$*, and the total shortfall is* $\sum_{v_i<0} v_i$*, these statistics can easily be calculated with only slight modifications to Algorithm 4.*

Proposition 2. *(i) Algorithm 4 is data-oblivious and computes an approximation to the market values in an n n debt network using only* $(k+3)n^3 + 6n^2$ *multiplications and* n^2 *comparisons. (ii) If there are no defaulters, Algorithm 4 will converge to the* exact *solution, when* $k = 1$*.*

Looking at Algorithm 4, there are no conditionals, the two "for" loops (Steps 4 and 11) both have fixed bounds (n). The Steps 5, 6 and 12 require only matrix multiplication and are therefore naturally oblivious. Given the sub-algorithms FindFix and UpdateLambda are also oblivious, the entire algorithm is as well.

Algorithm 4 requires $O\left(kn^3\right)$ (secure) multiplications, because it makes n calls to FindFix, which executes k $n \quad n$ matrix multiplications (which takes $O\left(kn^2\right)$ operations using schoolbook

matrix arithmetic). These multiplications dominate the cost of executing Algorithm 4 securely, and motivate the need for reducing the complexity of the FindFix algorithm (which we do in Algorithm 5).

More generally, developing data-oblivious algorithms usually requires avoiding loops with data-dependent bounds or early-termination conditions. It also requires avoiding conditionals ("if" statements), or, as in Algorithm 3 evaluating both branches of a conditional and multiplexing the results. The complexity of this type of multiplexing grows exponentially if the conditionals are nested, e.g. in Gaussian Elimination where determining the next pivot location requires testing whether an element is zero.

5 Implementations with Real Data

Secure computation protocols can be extremely resource intensive, with high processing and bandwidth costs. In fact, the costs (real or perceived) of running MPC protocols has been a significant barrier to adoption. To assess the feasibility of securely computing financial analytics, we implemented and benchmarked Algorithm 4 using three different software frameworks for MPC: SCALE-MAMBA[11], EMP-toolkit (Wang et al. 2016), and MPyC (Schoenmakers 2019).

Each of these frameworks uses different cryptographic techniques and has different security guarantees which lead to different performance characteristics.

All the benchmarks reported in this section were collected on a single 15-core machine with 3.6GHz Intel Xeon Gold 5122 processors and 128GB of RAM. We ran all parties locally on this single machine which implies we had to simulate n instances of individual machines running the MPC protocols. While this limits the scale of the computations we are able to perform, it is important to note that in practice, the computational burden would be split across the parties of the network, each thus requiring a fraction of the computational power of our own setup.

5.1 Data Sources

The Bank for International Settlements (BIS) is an international institution that collects financial statistics from many countries. They provide consolidated quarterly statistics on foreign and domestic claims and liabilities. There are 25-30 reporting countries each quarter who provide statistics relating to a larger set of several hundred countries.

We use these statistics to model a financial network in the debt model (§3.1). In this setting,

[11] https://homes.esat.kuleuven.be/~nsmart/SCALE/

network participants are countries, cross-holdings are international claims on other participants, and cash reserves are total domestic capital/equity.

In order to provide the complete liability network, we limit network participants to only the reporting countries who provide statistics on their total assets and equity. We used data for quarter 2 of 2018 but our data parsing tools can be used for any time period. This left us with 21 countries, as shown in Figure 4. Cross-country liabilities and represented by arrows, with darker arrows representing higher debts. The node size represents the country's underlying capital.

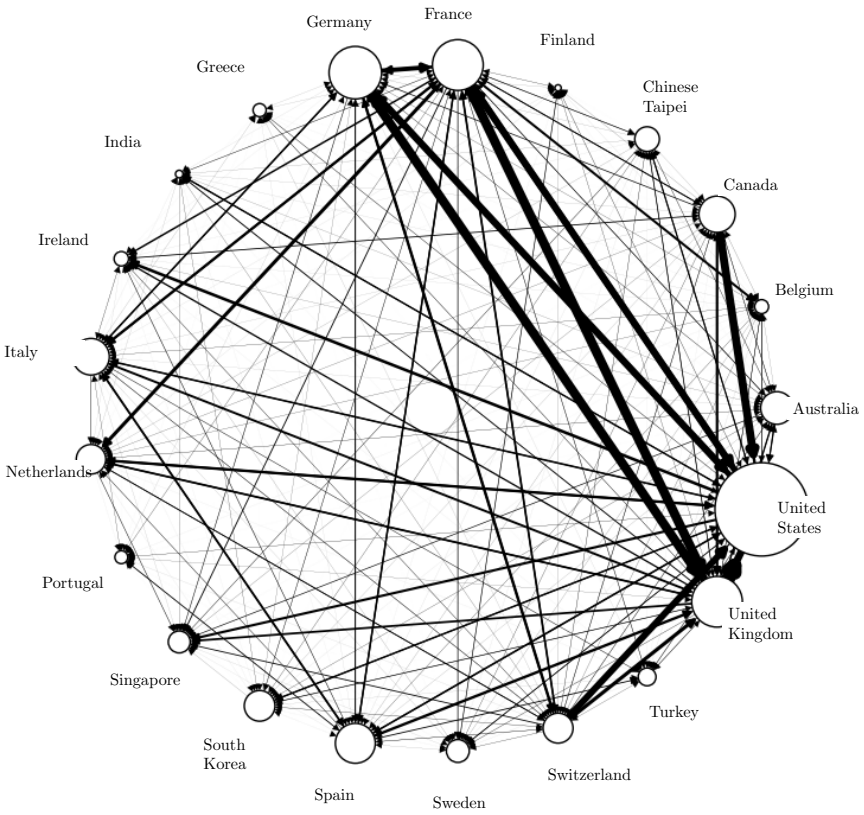


Figure 4: The full 21-country BIS dataset.

To model smaller networks, we worked with subsets of the BIS data set, and to model larger networks, we generated synthetic data. Note that a crucial feature of all secure computation algorithms is that the running times and memory usage depends only on the *size* of the network, and not the actual cross-holdings, since any dependence on cross-holdings would constitute a data leakage. Thus the estimates of running time and memory usage will be the same for *any* collection of real or synthetic data. In particular, this means our resource usage measurements are consistent across real and synthetic data as long as our generated data is the same size as the real data (*i.e.*, the values can be represented using 64-bit fixed point numbers, and the number of participants is the

same).

Table 1 shows the full 21-country network we extracted from the BIS data set. To test on smaller networks, we considered the subnetworks obtained by the considering only the first t banks in the table for t 21. The equilibrium market values are computed using the non privacy-preserving Algorithm 1. The privacy-preserving market values are computed using our data-oblivious Algorithm 4. There is only one defaulter in this network, so it is not surprising that the error term is zero (Proposition 2(ii) shows that if there are *no* defaulters the error will always be 0).

Table 1: Equilibrium market values of the 21-country BIS data set. Values are in millions of USD.

Name	**Reserves**	**Exact Market Value**	**Privacy-Preserving Market Value**	**Error**
Australia	196,291	324,605	324,605	0
Belgium	35,992	110,776	110,776	0
Canada	242,895	802,207	802,207	0
Chinese Taipei	112,044	146,500	146,500	0
Finland	7,231	21,196	21,196	0
France	476,060	1,726,679	1,726,679	0
Germany	509,167	1,367,904	1,367,904	0
Greece	31,955	37,005	37,005	0
India	10,622	99,027	99,027	0
Ireland	38,679	38,483	38,483	0
Italy	247,149	529,981	529,981	0
Netherlands	160,377	376,396	376,396	0
Portugal	30,689	34,555	34,555	0
Singapore	88,725	79,733	79,733	0
South Korea	173,528	213,149	213,149	0
Spain	293,666	573,661	573,661	0
Sweden	96,427	157,070	157,070	0
Switzerland	165,399	1,002,046	1,002,046	0
Turkey	58,544	732,388	732,388	0
United Kingdom	473,852	2,010,311	2,010,311	0
United States	1,605,782	2,429,208	2,429,208	0

In addition to the BIS data set, in some of our implementations we also considered the 6-country network consisting of France, Germany, Greece, Italy, Portugal and Spain analyzed in Elliott et al. (2014).[12] See a visualization in Figure 5.

[12] Although Elliott et al. (2014) analyzed this as an *equity* network, the underlying data consisted of *debts*. In that work, they converted the underlying debt network to an equity network by assuming each country retained 2/3 self-holdings, and normalizing the outstanding debts.

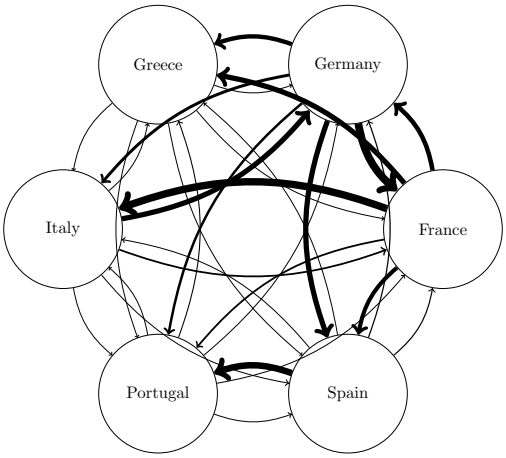
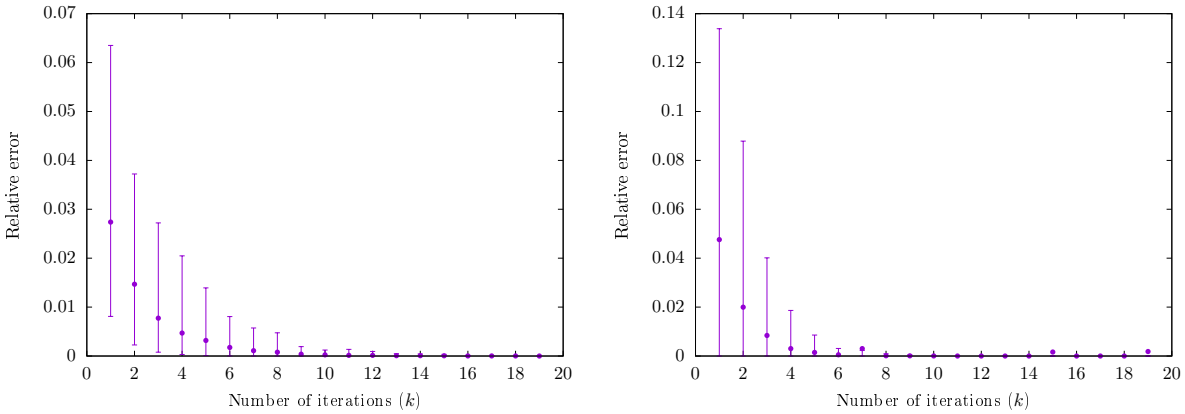


Figure 5: Six-country network (Elliott et al. 2014).

5.2 Rate of Convergence of Algorithm 2

The MPC algorithms we develop in §4 are error-free, in the sense that the privacy-preserving adjustments do not introduce any bias in the final output, with one exception: Algorithm 2, which replaces matrix inversion, is only asymptotically exact.

Figure 6 shows the algorithm's relatively quick rate of convergence in two different types of networks. Figure 6a considers a set of (synthetic) random 20-bank networks, and shows how the error in calculating market values decreases as a function of k, the number of iterations in Algorithm 2. The networks were generated with L_{ij} for $i \neq j$ uniformly distributed in $[0, 1]$. The relative error was calculated as $\frac{k\vec{p}_{\text{approx}} \quad \vec{p}_{\text{true}}k_1}{k\vec{p}_{\text{true}}k_1}$. The error bars show the 95% confidence interval based on 200 samples.



(a) Random 20-bank network

(b) Six-country network (from Figure 5)

Figure 6: FindFix Algorithm 2 convergence over k.

Figure 6b shows the accuracy of the oblivious FindFix algorithm on the real-world data set

from Elliott et al. (2014) (as represented in Figure 5). In this 6-country network, all the countries are solvent, and in this particular case, the FindFix algorithm converges with *no* error (see Proposition 2(ii)). In this setting, we tested the stability of our algorithm by simulating stress tests, reducing each country's underlying assets by a random variable, and re-running Algorithm 4 while keeping the underlying network fixed. shows the relative error of the market-clearing vector returned by the oblivious algorithm (Algorithm 4) compared to the ground truth for increasing number of iterations, k. The bars represent a 95% confidence interval. In all tests, the inter-country debts remained fixed, and only the countries' self-holdings dropped.

In both cases, setting $k = 10$ effectively reduces the error to 0.

5.3 Implementing and benchmarking Algorithm 4

Algorithm 4 shows how to compute market values in a data-oblivious manner, using only simple operations (addition, multiplication and comparisons).

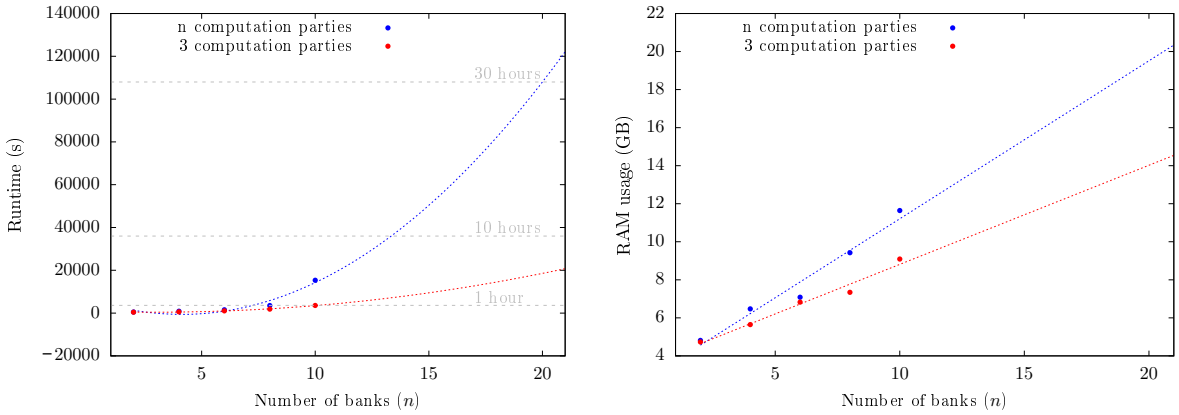
All three MPC frameworks we use provide a method for executing these basic operations securely, and we implemented Algorithm 4 in each of them in order to compute market values in a privacy preserving way. For each framework, we describe the guarantees and potential use-cases. We benchmark the time and memory resources required to run a computation on networks that can be handled by our single machine, and project expected resource requirements for larger networks that would be run in a distributed fashion.

SCALE-MAMBA SCALE-MAMBA is an open-source software system developed at KU Leuven. It implements a custom multi-party computation algorithm based on linear-secret sharing (Bendlin et al. 2011, Damgård et al. 2012, Nielsen et al. 2012) that is secure against a malicious adversary for an honest majority of participants. The system runs in three parts: the compilation phase consumes a user-described program to produce an executable format, the offline phase generates cryptographic material that is independent of the participants' input data (as described in §B.1.4. Finally, the online phase executes the computation.

The offline phase is the most time- and resource-intensive. It generates and validates several forms of cryptographic material in the form of samples of correlated random variables. This correlated randomness is independent of the participants' private inputs, and can be generated and stored at any point prior to executing the protocol. These values are then used to mask or "blind" secret values during the execution of the protocol. See Algorithm 9 in §B.1.1 for a full description. Since this offline phase is independent of the participants' data, it can be it can be run before the

participants know their (private) inputs. For example, the offline phase could be run overnight while markets are closed, then when the markets open, the (faster) online phase could be executed using up-to-the-minute market data.

In Figure 7, we show total resource use required *per party* to compute market values using Algorithm 4. These resources are for the cumulative computation, including both online and offline phases. The blue line represents the original setting with n computational parties, while the red line represents the more efficient "outsourcing" setting described in Section 2.4, whereby computations for the entire network were outsourced to a smaller 3-party set.



(a) Wall-clock time.

(b) RAM usage.

Figure 7: Per-party resources required to run Algorithms 4 using the SCALE-MAMBA MPC framework. Given we chose to run all parties on a single machine, we benchmarked our implementation for smaller numbers of parties and extrapolated to determine resources required for a larger number of parties.

SCALE-MAMBA provides extremely strong security guarantees (security against malicious adversaries), however, the processor and memory requirements are significant. Our single machine could not handle the full 21-party BIS dataset due to memory limitations, but this should not be an issue in practice given the computational burden would be split between the nodes in the network. We estimate that each machine would require around 18GB of RAM per party, which is highly feasible. Unfortunately, the running time is high – with an estimate of over 30 hours of computation to compute market values in a 21-bank network. Although this is, perhaps, acceptable in some situations (e.g. computing weekly stress tests), it does not seem likely to scale to much larger networks without significant algorithmic advances. The 3-party reduction affords meaningful efficiency gains (40% reduction in runtime), but is still within the same order of magnitude. The next implementation (using MPyC) *is* orders of magnitude faster, but uses a weaker security model.

MPyC This framework is secure against a semi-honest adversary when there is an honest majority. This is a weaker security setting than the other two frameworks and it does not require a pre-processing stage to generate cryptographic material. MPyC is implemented as a Python package and runs as interpreted language.

MPyC includes an implementation of a special-purpose secure protocol for solving linear equations of the form $\mathbf{A}\vec{x} = \vec{b}$ for an unknown vector $\vec{x}$ (Blom et al. 2019). The LinSol algorithm in Blom et al. (2019)[Protocol 4] securely solves linear equations using integer operations only. To take advantage of this, we develop Algorithm 5 below.

Algorithm 5 FindFix v.II

Input: $\mathbf{A}, \vec{b}$
$\bar{\mathbf{A}} = \mathbf{I} \quad \mathbf{A}$
$(\text{adj } \mathbf{A})\vec{b}, \det \mathbf{A} = \mathsf{LinSol}(\bar{\mathbf{A}}, \vec{b})$
$\vec{p} = (\text{adj } \mathbf{A})\vec{b} / \det \mathbf{A}$
return $\vec{p}$

LinSol( ) computes solutions directly, *without* resorting to iterative methods, but only works with *integer* inputs. Our setting requires real-valued inputs. To convert between real-valued (fixed-point) values and integer values requires, we follow the method proposed in Blom et al. (2019). We choose a scaling factor α, left-shift[13] all values in $\mathbf{A}$ and $\vec{b}$ by α and truncate into an integer. If α is too small, information about the fractional part of the value is lost. If α is large, we must use more storage to store each integer, which hurts efficiency. In our implementation, we represent initial data as 64-bit fixed point numbers with a 32-bit fractional part and set parameter $\alpha = 15$. We use large (64-bit) floating-point representations which ensures that we do not encounter errors related to truncation or overflow on realistic inputs.

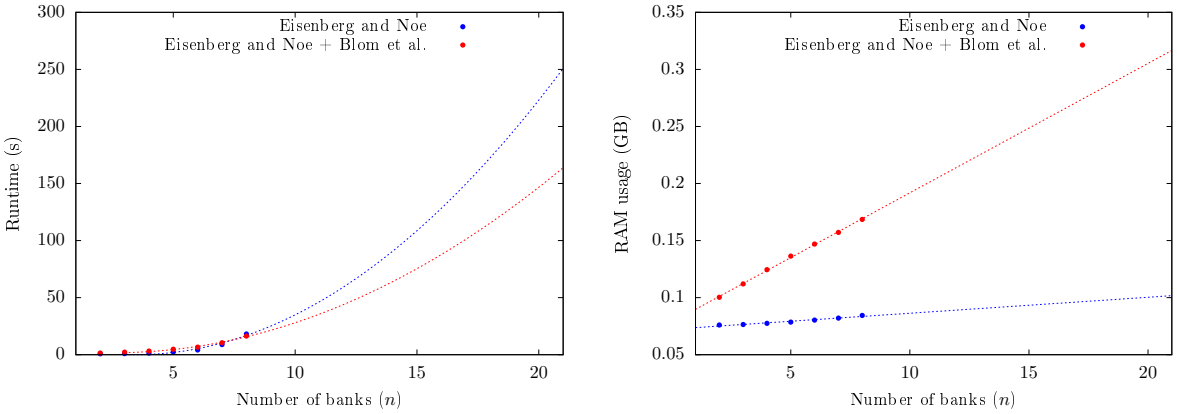
An additional source of overhead cost in Algorithm 5 returns the adjugate matrix (e.g. the inverse with each entry multiplied by the determinant). To compute the final market values, we eventually need to divide out the determinant. Division is inherently slow and also requires more space (e.g. another type conversion to larger fixed-point numbers).

We implemented both models using MPyC and show the results in Figure 8. As expected, the iterative algorithm is slower with larger numbers of parties (Figure 8a), however, it also has a lower memory overhead (Figure 8b). Our implementation is also one of the first to show the concrete efficiency gains from the special-purpose secure matrix inversion algorithm of Blom et al. (2019).

Using the MPyC framework, the computation on the full 21-bank network is only expected to

[13] *i.e.*, multiply by 2^{α}.

take about four minutes, which makes it extremely feasible for networks of this size, and much faster than the SCALE-MAMBA toolkit. However, MPyC only provides security against *passive* ("semi-honest") adversaries. By contrast, SCALE-MAMBA (and EMP, which we discuss next) provide security against *malicious* adversaries, and supporting this more robust security model adds to the complexity of the underlying secure computation protocol.



(a) Wall-clock time.

(b) RAM usage.

Figure 8: Total resources required to run Algorithm 4 using the MPyC framework. We benchmarked our implementation for up to 15 parties on the BIS data described in Section 5.1.

EMP-toolkit EMP-toolkit includes implementations of several garbled-circuit-based MPC protocols; we used the multi-party protocol secure against a dishonest majority of maliciously corrupted adversaries. Like SCALE-MAMBA, the algorithm runs in two phases: given a circuit describing the function, the parties run two preliminary protocols to generate cryptographic material. Then they run the secure computation protocol to evaluate the circuit.

EMP-toolkit works in the boolean circuit model, first converting the desired computation into a circuit consisting of AND, XOR and NOT gates, and then securely executing the circuit. Algorithm 4 requires matrix operations over the reals, and their corresponding boolean circuits are extremely complex, requiring hundreds of millions of gates. To illustrate the complexity of the secure computation algorithm we used EMP toolkit to generate *boolean circuits* computing Algorithm 4. The circuit sizes (as a function of the number of iterations, k) are plotted in Figure 9.

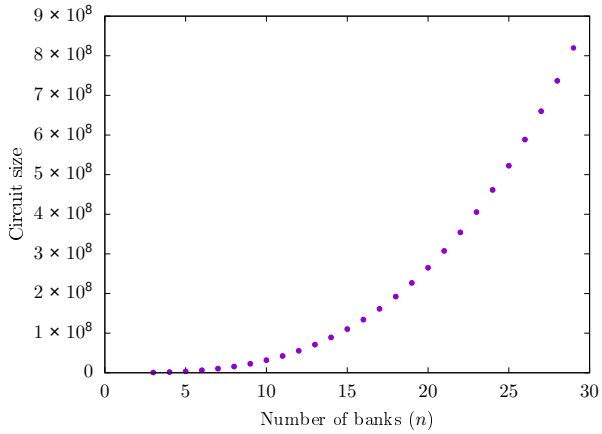


Figure 9: Circuit sizes for the clearing algorithm (Algorithm 4) as implemented in EMP (Wang et al. 2017). The plot shows the number of boolean gates required to securely execute the algorithm with $k = 10$. These circuits are large, requiring over 800 million gates when $n = 30$.

The circuit size gives an indication of the complexity of the computation that is independent from the specific computing hardware and networking architecture.

Unfortunately, we were unable to *execute* these circuits using EMP toolkit, however, previous work Kreuter et al. (2012) showed how to execute a garbled-circuit protocol in the malicious security model extremely efficiently. They report executing a circuit with 5.9 billion gates in 14,700 seconds. Extrapolating to our scenario (where our circuits have only 800 million gates), the computation time on their setup would only be around 20 seconds.

Summary Our experiments show that although the calculations are computationally intensive, they are *feasible* for small networks (n 20) on commodity hardware. If performance were a consideration, real-world deployments could dramatically increase the run-times with more powerful hardware and a more highly optimized software implementation.

It is also interesting to note difference in runtimes between SCALE-MAMBA and MPyC. The two frameworks use extremely different MPC back-ends so it is hard to pinpoint a single cause of the performance differences. Nevertheless, it seems likely that providing security against malicious adversaries (as in SCALE-MAMBA) is a primary cause of its slower performance. If this is the case, it is an interesting question whether practitioners (e.g. banks or regulators) feel the need to provide security against malicious adversaries, or whether security against semi-honest adversaries is sufficient.

6 Discussion, Limitations and Conclusion

6.1 Equity Model of Elliott et al. (2014)

For robustness, beyond the debt model of Eisenberg and Noe (2001), we have implemented and benchmarked the market-value calculations for the equity cross-holdings model of Elliott et al. (2014). To deal with privacy preservation in this case, we develop Algorithm 6, which is slightly simpler than Algorithm 4. As a result, the performance of the equity algorithm is slightly better than that of the debt algorithm. The equity model, algorithm and our implementation are described in detail in Appendix A. We emphasize that our methodology can be applied to this, and even more complex network models, such as those described in Appendix A.4.

6.2 Feasibility and Future Work

Although prior work has suggested the use of MPC for privacy-preserving financial computations (Abbe et al. 2012, Cai and Kou 2019), these works only considered generic protocols (e.g. sums, variances, moments, linear inference) rather than more complex protocols (e.g. market-values) from the literature. The one prior work that develops MPC for financial networks (Narayan et al. 2014) provides weaker security guarantees than ours, and does not analyze the convergence of their algorithms. Our work is the first to provide strong security and convergence guarantees and to provide benchmarks on real-world data. To demonstrate feasibility, we have implemented two different market-value computations, one in the debt model, and one in the equity model, using three different secure multiparty computation frameworks.

Our implementations (which we provide as open-source Docker images) show that these secure computations are indeed feasible for realistic networks. In terms of efficiency, we claim that existing tools are sufficient for use in some applications, including computing market values for small networks. We estimate SCALE-MAMBA requires less than 20GB of RAM and around 30 hours per party to compute market values of a network with 21 participants. This is a relatively low cost for a small group of banks who wish to run monthly risk assessments amongst themselves. There are a variety of known optimizations that could improve our results as well, such as porting Algorithm 5 to SCALE-MAMBA or EMP-toolkit and carefully tuning software parameters to optimize the computation. At a higher level, using special-purpose MPC protocols designed for highly efficient operations over real-valued matrices could dramatically improve runtimes.

The software tools we use here were developed as academic projects and are not, generally speaking, production quality code. Despite this, they demonstrate the *feasibility* of this approach.

In a real application setting, a coalition of financial institutions could fund the development an open-source, production-quality implementation. Coupling this with a moderate investment in hardware, we believe that these protocols could easily see significant performance improvements over our prototype implementations. This investment may well be worth making, as the development of such a tool would enable widespread adoption of secure computation.

Our work shows that key calculations in financial networks can be calculated in a privacy-preserving manner. Our work also highlights the key variables that affect the performance. Thus, any coalition of institutions considering this type of system needs to ask (1) How many institutions want to participate in the calculation (using secret-sharing, the number of institutions providing data can be larger than the number of institutions actively participating in the computation, see Section 2.4), (2) Do the institutions need security against malicious adversaries, or is semi-honest security sufficient? (3) How often do the calculations need to be run (e.g. is a running time of 6 hours acceptable)?

6.3 Limitations

Second-order inference: In this work, we focus on the question of securely computing market values, and financial stress-tests, without a trusted third party, and without revealing the stakeholders' underlying data. Like extant literature in this space, we do not address the scenario that the *result* of the calculation (e.g. the market values of the institutions, the number of institutions that fail, or the total financial shortfall) could reveal sensitive information about the underlying institutions. In principle, one could use secure computation to implement a differentially private stress-test to achieve both privacy against an adversary monitoring network traffic, and an adversary performing inference on the result.

Garbage-in-garbage-out problem: Our algorithms do not ensure that the participants behave *truthfully*. In fact, MPC protocols in general can only enforce "truthful" behavior if such behavior can formally (mathematically) defined. For example, our protocol could be easily extended to enforce simple consistency checks (e.g., in the debt model if i reports a debt to j, then j should report the same debt from i), but without a mathematical definition of "truth" for each participants private inputs, the protocol cannot enforce that the participants behave truthfully. Of course, if participants provide incorrect inputs, the calculated market values will also be incorrect.

Scalability: Our implementations show that these types of computations can easily scale to dozens (or even hundreds) of participants. Unfortunately, the communication costs of the underlying secure computation algorithms scale *quadratically* with the number of participants, so it is unlikely

that this architecture, in its current form, can scale to *thousands* of participants. To address settings with more participants, it is common to change the network architecture, separating the number of data owners (*i.e.*, the number of banks) from the number of computation parties (e.g. a subset of larger banks, or outside agencies). This allows us to increase the number of banks without increasing the number of computational parties. Figure 7 shows that running Algorithm 4 with three parties instead of 21 reduces the runtime by about 40%. Unfortunately, this reduces security in the sense that all privacy is lost if *all* the computation parties collude (even if the majority of the *input* parties do not collude).

References

Abbe, E. A., A. E. Khandani, A. W. Lo. 2012. Privacy-Preserving Methods for Sharing Financial Risk Exposures. *American Economic Review* **102**(3) 65–70. doi:10.1257/aer.102.3.65. URL http://dx.doi.org/10.1257/aer.102.3.65.

Acemoglu, D., A. Ozdaglar, A. Tahbaz-Salehi. 2015. Systemic risk and stability in financial networks. *American Economic Review* **2015** 564 – 608. doi:10.3386/w18727. URL http://www.nber.org/papers/w18727.

Allen, F., D. Gale. 1998. Optimal Financial Crises. *The Journal of Finance* **53**(4) 1245–1284. doi:10.1111/0022-1082.00052. URL http://dx.doi.org/10.1111/0022-1082.00052.

Allen, F., D. Gale. 2000. Financial Contagion. *Journal of Political Economy* **108**(1) 1+. doi:10.1086/262109. URL http://dx.doi.org/10.1086/262109.

Aly, A., M. Keller, D. Rotaru, P. Scholl, N. P. Smart, T. Wood. 2019. Scale-mamba. https://homes.esat.kuleuven.be/ nsmart/SCALE/.

Babich, V., S. Marinesi, G. Tsoukalas. 2020. Does crowdfunding benefit entrepreneurs and venture capital investors? *Manufacturing & Service Operations Management* .

Beaver, D. 1991. Efficient multiparty protocols using circuit randomization. *CRYPTO*.

Beaver, D., S. Micali, P. Rogaway. 1990. The round complexity of secure protocols. *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. 503–513.

Beimel, A. 2011. Secret-Sharing Schemes: A Survey. Y. Chee, Z. Guo, S. Ling, F. Shao, Y. Tang, H. Wang, C. Xing, eds., *Coding and Cryptology*, *Lecture Notes in Computer Science*, vol. 6639. Springer Berlin Heidelberg, 11–46. doi:10.1007/978-3-642-20901-7_2. URL http://dx.doi.org/10.1007/978-3-642-20901-7_2.

Belavina, E., S. Marinesi, G. Tsoukalas. 2020. Rethinking crowdfunding platform design: mechanisms to deter misconduct and improve efficiency. *Management Science* .

Bellare, M., V. T. Hoang, P. Rogaway. 2012a. Adaptively secure garbling with applications to one-time programs and secure outsourcing. *ASIACRYPT*. Springer, 134–153.

Bellare, M., V. T. Hoang, P. Rogaway. 2012b. Foundations of garbled circuits. *CCS*. 784–796.

Ben-Or, M., S. Goldwasser, A. Wigderson. 1988. Completeness theorems for non-cryptographic fault-tolerant distributed computation. *STOC*. ACM, New York, NY, USA, 1–10. URL http://doi.acm.org/10.1145/62212.62213.

Bendlin, R., I. Damgård, C. Orlandi, S. Zakarias. 2011. Semi-homomorphic encryption and multiparty computation. *EUROCRYPT '11*. 169–188. doi:10.1007/978-3-642-20465-4_11. URL https://doi.org/10.1007/978-3-642-20465-4_11.

Bestavros, A., A. Lapets, M. Varia. 2017. User-centric distributed solutions for privacy-preserving analytics. *Communications of the ACM* **60**(2) 37–39.

Biais, B., C. Bisiere, M. Bouvard, C. Casamatta. 2019. The blockchain folk theorem. *The Review of Financial Studies* **32**(5) 1662–1715.

Blom, F., N. J. Bouman, B. Schoenmakers, N. de Vreede. 2019. Efficient secure ridge regression from randomized gaussian elimination. IACR ePrint 2019/773.

Blume, L., D. Easley, J. Kleinberg, R. Kleinberg, E. Tardos. 2011. Which Networks are Least Susceptible to Cascading Failures? *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*. IEEE, 393–402. doi:10.1109/focs.2011.38. URL http://dx.doi.org/10.1109/focs.2011.38.

Bogdanov, D., M. Jõemets, S. Siim, M. Vaht. 2016. Privacy-preserving tax fraud detection in the cloud with realistic data volumes. Tech. rep., Technical Report T-4-24. Cybernetica AS, http://research. cyber. ee.

Boyle, E., K.-M. Chung, R. Pass. 2015. Large-scale secure computation: Multi-party computation for (parallel) RAM programs. *Annual Cryptology Conference*. Springer, 742–762.

Brioschi, F., L. Buzzacchi, M. G. Colombo. 1989. Risk capital financing and the separation of ownership and control in business groups. *Journal of Banking & Finance* **13**(4-5) 747–772. doi:10.1016/0378-4266(89)90040-x. URL http://dx.doi.org/10.1016/0378-4266(89)90040-x.

Cai, N., S. Kou. 2019. Econometrics with privacy preservation. *Operations Research* **67**(4) 905–926.

Chakraborty, S., R. Swinney. 2020. Signaling to the crowd: Private quality information and rewards-based crowdfunding. *Manufacturing & Service Operations Management* .

Chaum, D., C. Crépeau, I. Damgård. 1988. Multiparty Unconditionally Secure Protocols. *STOC*. 11–19. URL http://dl.acm.org/citation.cfm?id=62214.

Chod, J., E. Lyandres. 2020. A theory of icos: Diversification, agency, and information asymmetry. *Management Science* .

Chod, J., N. Trichakis, G. Tsoukalas, H. Aspegren, M. Weber. 2020. On the financing benefits of supply chain transparency and blockchain adoption. *Management Science* .

Cong, L. W., Y. Li, N. Wang. 2020. Tokenomics: Dynamic adoption and valuation. Tech. rep., National Bureau of Economic Research.

Damgård, I., V. Pastro, N. Smart, S. Zakarias. 2012. Multiparty computation from somewhat homomorphic encryption. R. Safavi-Naini, R. Canetti, eds., *Advances in Cryptology – CRYPTO 2012*. Springer Berlin Heidelberg, Berlin, Heidelberg, 643–662.

Demmler, D., T. Schneider, M. Zohner. 2015. ABY - a framework for efficient mixed-protocol secure two-party computation. *NDSS*. doi:10.14722/ndss.2015.23113.

DFAST. 2019. Dodd-Frank Act Stress Tests (DFAST). https://www.fhfa.gov/SupervisionRegulation/DoddFrankActStressTes

Dodd-Frank. 2010. Dodd-Frank wall street reform and consumer protection act. Public Law 111-203, US Statutes at Large.

Dwork, C. 2011. Differential privacy. *Encyclopedia of Cryptography and Security* 338–340.

Economist, T. 2014. Note to future self. *The Economist* .

Eisenberg, L., T. H. Noe. 2001. Systemic Risk in Financial Systems. *Management Science* **47**(2) 236–249. URL http://www.jstor.org/stable/2661572.

Elliott, M., B. Golub, M. O. Jackson. 2014. Financial networks and contagion. *American Economic Review* **104**(10) 3115–53. URL https://www.aeaweb.org/articles?id=10.1257/aer.104.10.3115.

Elsinger, H. 2011. Financial networks, cross holdings, and limited liability. Tech. Rep. 156, Oesterreichische Nationalbank.

Even, S., O. Goldreich, A. Lempel. 1985. A randomized protocol for signing contracts. *Communications of the ACM* **28**(6) 637–647.

Fedenia, M., J. E. Hodder, A. J. Triantis. 1994. Cross-holdings: estimation issues, biases, and distortions. *Review of Financial Studies* **7**(1) 61–96. doi:10.1093/rfs/7.1.61. URL http://dx.doi.org/10.1093/rfs/7.1.61.

Feinstein, Z., W. Pang, B. Rudloff, E. Schaanning, S. Sturm, M. Wildman. 2018. Sensitivity of the Eisenberg–Noe clearing vector to individual interbank liabilities. *SIAM Journal on Financial Mathematics* **9**(4) 1286–1325.

Flood, M., J. Katz, S. Ong, A. Smith. 2013. Cryptography and the Economics of Supervisory Information:Balancing Transparency and Condentiality. Tech. Rep. 0011, Office of Financial Research. URL http://www.treasury.gov/initiatives/ofr/research/Documents/OFRwp0011_FloodKatzOngSmith_CryptographyAndTheEconomicsOfSupervisoryInformation.pdf.

Franz, M., A. Holzer, S. Katzenbeisser, C. Schallhart, H. Veith. 2014. CBMC-GC: An ANSI C Compiler for Secure Two-Party Computations. A. Cohen, ed., *Compiler Construction*, *Lecture Notes in Computer Science*, vol. 8409. Springer Berlin Heidelberg, 244–249. doi:10.1007/978-3-642-54807-9_15. URL http://dx.doi.org/10.1007/978-3-642-54807-9_15.

French, K. R., J. M. Poterba. 1991. Were Japanese stock prices too high? *Journal of Financial*

Economics **29**(2) 337–363. doi:10.1016/0304-405x(91)90006-6. URL http://dx.doi.org/10.1016/0304-405x(91)90006-6.

Gai, P., S. Kapadia. 2010. Contagion in financial networks. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science* **466**(2120) 2401–2423. doi:10.1098/rspa.2009.0410. URL http://dx.doi.org/10.1098/rspa.2009.0410.

Gan, R. J., G. Tsoukalas, S. Netessine. 2020. Initial coin offerings, speculators, and asset tokenization. *Management Science* .

Garg, S., D. Gupta, P. Miao, O. Pandey. 2016. Secure multiparty RAM computation in constant rounds. *Theory of Cryptography Conference*. Springer, 491–520.

Goldreich, O., S. Micali, A. Wigderson. 1987. How to Play any Mental Game. *STOC*. 218–229. doi:10.1145/28395.28420. URL http://doi.acm.org/10.1145/28395.28420.

Gouriéroux, C., J. C. Héam, A. Monfort. 2012. Bilateral exposures and systemic solvency risk Expositions bilatérales et risque systémique pour la solvabilité . *Canadian Journal of Economics/Revue canadienne d'économique* **45**(4) 1273–1309. doi:10.1111/j.1540-5982.2012.01750.x. URL http://dx.doi.org/10.1111/j.1540-5982.2012.01750.x.

Hastings, M., B. H. Falk, D. Noble, S. Zdancewic. 2019. SoK: General purpose compilers for secure multi-party computation. *S&P*.

Hemenway, B., S. Khanna. 2016. Sensitivity and computational complexity in financial networks. *Algorithmic Finance* **5**(3-4) 95–110.

Hinzen, F. J., K. John, F. Saleh. 2019. Bitcoin's fatal flaw: The limited adoption problem. *NYU Stern School of Business* .

Hirahara, S., I. C. Oliveira, R. Santhanam. 2018. Np-hardness of minimum circuit size problem for or-and-mod circuits. ECC TR-18-030. URL https://eccc.weizmann.ac.il/report/2018/030/.

Jackson, M. O., A. Pernoud. 2019. What makes financial networks special? distorted investment incentives, regulation, and systemic risk measurement. SSRN. URL https://ssrn.com/abstract=3311839.

Johnson, C. R. 1982. Inverse M-matrices. *Linear Algebra and its Applications* **47** 195–216. doi:10.1016/0024-3795(82)90238-5. URL http://dx.doi.org/10.1016/0024-3795(82)90238-5.

Keller, M., E. Orsini, P. Scholl. 2016. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 830–842.

Keller, M., V. Pastro, D. Rotaru. 2018. Overdrive: making SPDZ great again. *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 158–189.

Kreuter, B., A. Shelat, C.-H. Shen. 2012. Billion-gate secure computation with malicious adversaries. *Presented as part of the 21st fUSENIXg Security Symposium (fUSENIXg Security 12)*. 285–300.

Lapets, A., F. Jansen, K. D. Albab, R. Issa, L. Qin, M. Varia, A. Bestavros. 2018. Accessible privacy-

preserving web-based data analysis for assessing and addressing economic inequalities. *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies*. ACM, 48.

Liu, C., Y. Huang, E. Shi, J. Katz, M. Hicks. 2014. Automating efficient RAM-model secure computation. *2014 IEEE Symposium on Security and Privacy*. IEEE, 623–638.

Lu, S., R. Ostrovsky. 2013. How to garble RAM programs? *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 719–734.

Merkel, D. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux journal* **2014**(239) 2.

Mitchell, J. C., J. Zimmerman. 2014. Data-oblivious data structures. *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Mohassel, P. 2011. Efficient and secure delegation of linear algebra. IACR ePrint 2011/605.

Morris, S. 2000. Contagion. *Review of Economic Studies* **67**(1) 57–78. doi:10.1111/1467-937X.00121. URL `http://www.princeton.edu/{~}smorris/pdfs/paper{_}24{_}Contagion.pdf`.

Narayan, A., A. Papadimitriou, A. Haeberlen. 2014. Compute globally, act locally: Protecting federated systems from systemic threats. *10th Workshop on Hot Topics in System Dependability (HotDep 14)*.

Nielsen, J. B., P. S. Nordholt, C. Orlandi, S. S. Burra. 2012. A new approach to practical active-secure two-party computation. *Advances in Cryptology–CRYPTO 2012*. Springer, 681–700.

Rosu, I., F. Saleh. 2020. Evolution of shares in a proof-of-stake cryptocurrency. *Management Science* .

Sangers, A., M. van Heesch, T. Attema, T. Veugen, M. Wiggerman, J. Veldsink, O. Bloemen, D. Worm. 2019. Secure multiparty pagerank algorithm for collaborative fraud detection. *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*. 605–623.

Schoenmakers, B. 2019. MPyC: secure multiparty computation in Python. Github. URL `https://github.com/lschoe/mpyc`.

Schwenkler, G., H. Zheng. 2019. The network of firms implied by the news. Available at SSRN 3320859.

Songhori, E. M., S. U. Hussain, A.-R. Sadeghi, T. Schneider, F. Koushanfar. 2015. TinyGarble: Highly Compressed and Scalable Sequential Garbled Circuits. *IEEE S & P*. URL `http://thomaschneider.de/papers/SHSSK15.pdf`.

Sternberg, S. 2010. *Dynamical systems*. Courier Corporation.

Tsoukalas, G., B. H. Falk. 2020. Token-weighted crowdsourcing. *Management Science* .

Tsoukalas, G., S. Marinesi, V. Babich. 2019. Updating the crowdfunding narrative. *Wharton Public Policy Initiative* .

Wang, X., A. J. Malozemoff, J. Katz. 2016. EMP-toolkit: Efficient MultiParty computation toolkit. `https://github.com/emp-toolkit`.

Wang, X., S. Ranellucci, J. Katz. 2017. Global-scale secure multiparty computation. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 39–56.

Wang, X. S., C. Liu, K. Nayak, Y. Huang, E. Shi. 2015. Oblivm: A programming framework for secure computation. *IEEE Symposium on Security and Privacy (S & P)*. URL http://www.cs.umd.edu/~{}elaine/docs/oblivm.pdf.

Willoughby, R. A. 1977. The inverse M-matrix problem. *Linear Algebra and its Applications* **18**(1) 75–94. doi:10.1016/0024-3795(77)90081-7. URL http://dx.doi.org/10.1016/0024-3795(77)90081-7.

Yao, A. 1982. Protocols for Secure Computations (Extended Abstract). *FOCS '82*. 160–164. doi:10.1109/SFCS.1982.88. URL http://dx.doi.org/10.1109/SFCS.1982.88.

Yao, A. 1986. How to Generate and Exchange Secrets. *FOCS '86*. 162–167. doi:10.1109/SFCS.1986.25. URL http://portal.acm.org/citation.cfm?id=1382944.

Zahur, S., D. Evans. 2015. Obliv-C: A language for extensible data-oblivious computation. IACR Cryptology ePrint Archive 2015/1153. URL https://eprint.iacr.org/2015/1153.

Appendix

A Privacy Preservation in Alternative Network Models

A.1 Equity Cross-Holdings Model

In this section, we develop privacy-preserving algorithms in an alternative model of financial networks, in which institutions to hold shares of other institutions rather than debt contracts (Elliott et al. 2014). The algorithms rely on the same methods as the ones developed for the debt model in Section 4, and the overall conclusion of this section is that the methods remain applicable in alternative network settings.

In this alternative model, each institution represents a node in the network and institutions are connected by proportional *cross-holdings*, which are represented as weighted, directed edges in the graph. An edge from institution i to institution j of weight p (with $0 \quad p \quad 1$) represents the statement that institution i owns a p-fraction of institution j. In addition to these proportional cross-holdings or shares, each bank can have some underlying assets, and these assets are the sole source of value in the network. These assets could represent tangible, physical assets or any investment the institution has made outside the network. This model can also be viewed as a flow, where at each time step money flows between institutions, and the total incoming flow to a node is divided among its outgoing edges according to their edge weights. With this intuition, the "market value" of a bank is the steady-state flow through that institution.

The possibility of cycles in the network (where institution a owns shares of b, who owns shares of c, who owns shares of a) means that the market value of institutions can be extremely sensitive to small changes in the cross-holdings (Hemenway and Khanna 2016). In fact, if institutions can have arbitrarily small self-ownership, then an ϵ-change in cross-holdings can have arbitrarily large impact on market values. Figure 10 illustrates this in a simple network with 4 banks. The banks all have self-holdings ("reserves") of r, and the cross-holding values reported in the figure. Focusing on Bank 2, one can see that this bank would need to know the value of ϵ to be able to determine its own market value. If Bank 2 has an outside asset with value 1, and no other banks have any external assets, one can readily compute the vector of market values in closed form.[14]

[14]Namely, the market value vector is given by

$$\vec{v} = \frac{1}{(\epsilon + r - 1)(1-r)^2 + 1} \begin{bmatrix} (\epsilon + r - 1)(r-1)r \\ r \\ r(1-r) \\ \epsilon(1-r) \end{bmatrix}. \tag{11}$$

In Figure 11, we show the final market value of each bank, assuming each bank retains $r = 10\%$ self-holdings. The key point is that the value of Bank 2 drops significantly from .37 to .1 as ϵ goes from 0 to r, yet Bank 2 has no direct knowledge of ϵ.

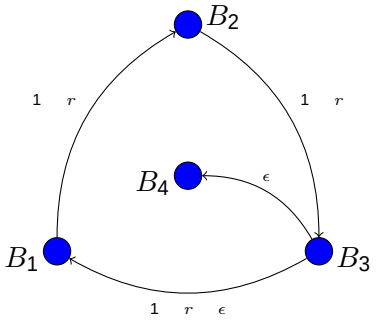


Figure 10: Four-bank network.

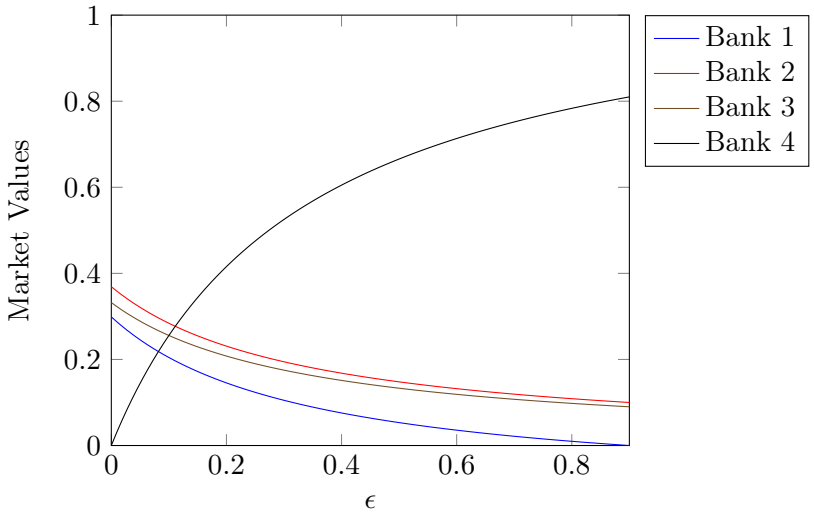


Figure 11: Market values of the banks in Figure 10.

The high sensitivity of market values to small changes in cross-holdings means that institutions must have a near perfect view of all other institutions' cross-holdings in order to calculate their own market value! In Section A.2, we give an Algorithm 6 for securely computing the approximate market values of each institution *without* requiring the institutions to share their cross-holdings (see Figure 12).

More formally, we will use the notation C_{ij} to denote the fraction of institution j owned by institution i. We will use n n matrix $\mathbf{C} = (C_{ij})$ to denote the cross-holdings within the network.

The sum of all market values is 1 (which is the total value of all external assets).

As in Elliott et al. (2014), we define $C_{ii} = 0$. The matrix $\mathbf{C}$ corresponds to the weighted, directed graph on n vertices that has an edge from j to i with weight C_{ij} whenever $C_{ij} > 0$. With this notation, $\sum_i C_{ij}$ represents the proportion of j that is owned by other institutions. We assume that $\sum_i C_{ij} < 1$, *i.e.*, that each institution remains some proportion of self-ownership. We define $\mathbf{S}$ to be the diagonal matrix with $S_{jj} \stackrel{\text{def}}{=} 1 \quad \sum_i C_{ij}$ to be the fraction of self-ownership of institution j. We will use D_i to denote the value of the assets of institution i, and the vector $\vec{D}$ to denote the vector of asset values.

This type of model leads to two different notions of valuations of an institution, the *equity* valuation, and the *market* valuation Brioschi et al. (1989). The equity valuation of institution i is denoted by

$$V_i = \underbrace{D_i}_{\text{Value of assets held by } i} + \underbrace{\sum_j C_{ij} V_j}_{\text{Values of institutions held by } i} \tag{12}$$

In matrix notation, this becomes

$$\vec{V} = \vec{D} + \mathbf{C}\vec{V},$$

which leads to the solution

$$\vec{V} = (\mathbf{I} \quad \mathbf{C})^{\ 1} \vec{D}. \tag{13}$$

As noted in Elliott et al. (2014), since the columns of $\mathbf{C}$ sum to strictly less than one, the matrix $\mathbf{I} \quad \mathbf{C}$ is guaranteed to be invertible. (Actually, $\mathbf{I} \quad \mathbf{C}$ is an M-Matrix, see Willoughby 1977, Johnson 1982).

It is relatively straightforward to see that the equity valuation significantly overcounts the underlying value of the system, since the value of each asset is counted towards its primary owner institution as well as the shareholders in that institution. Formally, this means $k\vec{V}k_1 \quad k\vec{D}k_1$. This property of the equity valuation is well-known (French and Poterba 1991, Fedenia et al. 1994).

The *market* valuation eliminates this overcounting, by scaling each institution's equity value by its percentage of self-ownership. Thus the market valuation of institution i is

$$\vec{v} = \mathbf{S}\vec{V} = \mathbf{S}(\mathbf{I} \quad \mathbf{C})^{\ 1} \vec{D}. \tag{14}$$

Obstacles for privacy preservation

From the standpoint of privacy, the main issue with Equation 14 is that every entry of $\vec{v}$ depends globally on the matrix $\mathbf{C}$. The ith institution may know its equity holdings (the ith row of $\mathbf{C}$)

and its shareholders (the ith column of $\mathbf{C}$), but its own market value (the ith coordinate of $\vec{v}$) is sensitive to the other entries of $\mathbf{C}$ as well, which may not be publicly known. How can one calculate the matrix inverse $(\mathbf{I} \quad \mathbf{C})^{\ 1}$ when the entries of $\mathbf{C}$ are private and held by different parties?

A.2 Privacy-Preserving Equity Model

As discussed in Section 3.2, secure computation algorithms must be *data oblivious*, *i.e.*, the algorithm's control flow cannot change based on the algorithm's (private) inputs. Thus we need to develop data-independent method for calculating the market values given in Equation 14.

As noted in Section B.1.3, matrix inversion is a particularly difficult task for secure computation. Fortunately, however, the inverse $(\mathbf{I} \quad \mathbf{C})^{\ 1}$ is well approximated by a power series $\lim_{t\ !\ 1} \sum_{k=0}^{t} \mathbf{C}^k \ !\ (\mathbf{I} \quad \mathbf{C})^{\ 1}$. In Lemma 1, we give explicit bounds on the rate of convergence.

Lemma 1 (Invertibility of $\mathbf{I} \quad \mathbf{C}$). *If* $s_{\min} \stackrel{def}{=} \min_i S_{ii} > 0$, *then*

$$(\mathbf{I} \quad \mathbf{C})^{\ 1} = \sum_{k=0}^{1} \mathbf{C}^k,$$

and for any $\vec{v}$ 6= $\vec{0}$,

$$\frac{k \sum_{k=0}^{N} \mathbf{C}^k \vec{v} \quad (\mathbf{I} \quad \mathbf{C})^{\ 1} \vec{v} k_1}{k(\mathbf{I} \quad \mathbf{C})^{\ 1} \vec{k}_1} < (1 \quad s_{\min})^{N+1}.$$

This leads to the following algorithm (Algorithm 6) for computing the market values in an equity network.

Algorithm 6 An iterative algorithm for calculating the market values in a equity cross-holdings network model.

```
Input: n  n cross-holdings matrix C
Input: n  1 asset vector D
Input: n  n diagonal self-holding matrix S
Initialize: n  n matrix A = 0
for i = 1, ..., k do
    A = C  A + I
end for

v = SAD
return v
```

Proposition 3. *If* $\mathbf{C}, \mathbf{S}, \vec{D}$ *is an equity-network with mininum self-holdings* $s_{\min} \stackrel{def}{=} \min_i S_{ii} > 0$,

then Algorithm 6 gives an approximate solution $\vec{v}$ to the true market values $\vec{v}_{\mathsf{true}}$ with

$$k\vec{v} \quad \vec{v}_{\mathsf{true}}k_1 < (1 \quad s_{\mathsf{min}})^{k+1}\vec{D}, \tag{15}$$

using only $kn^3 + n^2$ additions and $kn^3 + n^2 + n$ multiplications.

A.3 Implementations

We implement our Algorithm 6 using the same three MPC software packages described in the debt model of Section 5. For comparison, in some of the subsequent figures, we overlay results from the debt model on top of results from the equity model.

Four-Bank example of Figure 10 Figure 12 shows the banks' true market values as well as their approximate market values obtained from the privacy-preserving Algorithm 6, with $k = 10$. As in Figure 11, $r = .1$ (each bank retains 10% self-ownership), and only Bank 2 possesses any outside assets (these are normalized to have value 1).

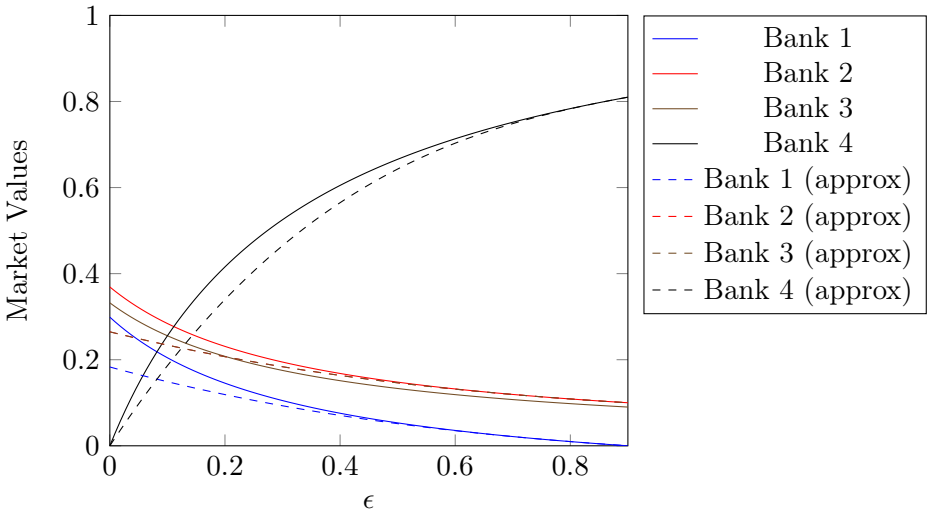


Figure 12: Four-bank Network of Figure 10. The solid lines are the true market values, and the dotted lines represent the approximate market values calculated by the iterative algorithm (Algorithm 6).

Convergence of Algorithm 6 For a set of random 20-bank networks, Figure 13 shows how the error in calculating market values decreases as a function of k, the number of iterations in Algorithm 6. The networks were generated with C_{ij} for $i \neq j$ uniformly distributed in $[0, 1]$, S_{ii} uniformly distributed in $[.1, .5]$, then rescaled so that the columns of $\mathbf{C} + \mathbf{S}$ sum to 1. The relative error was calculated as $\frac{k\vec{v}_{\text{approx}} \quad \vec{v}_{\text{true}}k_1}{k\vec{v}_{\text{true}}k_1}$. The error bars show the 95% confidence interval. Setting $k = 10$ (as we did in the rest of our implementations) effectively reduces the error to 0.

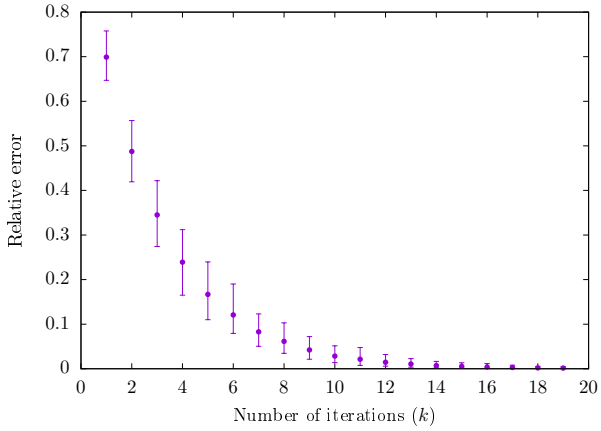
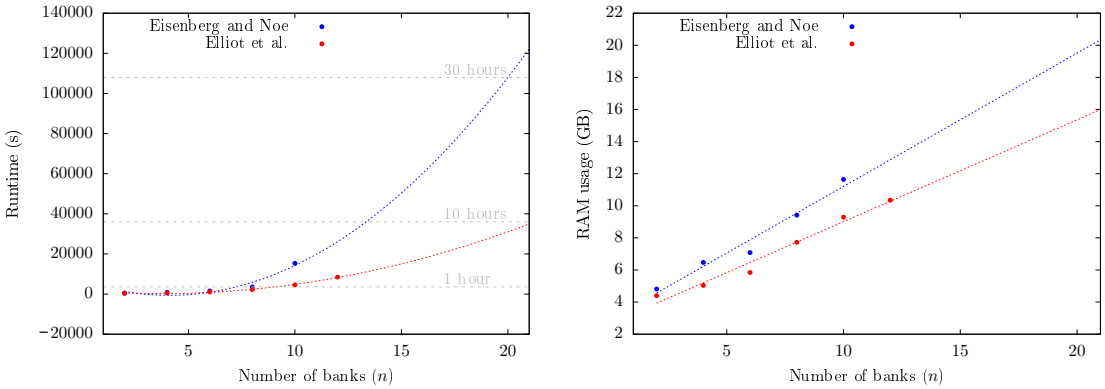


Figure 13: 20-bank networks

SCALE-MAMBA Figure 14 shows total resource use required *per party* to run our implementation in SCALE-MAMBA. These numbers are for the cumulative computation (including both online and offline phases). As expected, the equity model is less resource-intensive, since it only requires one iteration of the iterative fixed-point algorithm (Algorithm 2), while the debt model requires n in the number of banks.



(a) Wall-clock time.

(b) RAM usage.

Figure 14: Per-party resources required to run Algorithms 4 and 6 using the SCALE-MAMBA MPC framework. We benchmarked our implementation for smaller numbers of parties and extrapolated to determine resources required for a larger number of parties.

EMP-toolkit As a reminder, EMP-toolkit works in the boolean circuit model, first converting the desired computation into a circuit consisting of AND, XOR and NOT gates, and then securely executing the circuit. Algorithm 6 requires matrix operations over the reals, and their corresponding boolean circuits are extremely complex, requiring hundreds of millions of gates. To illustrate the complexity of the secure computation algorithm we plot the circuit sizes we required (as a function of the number of iterations, k) in Figure 15.

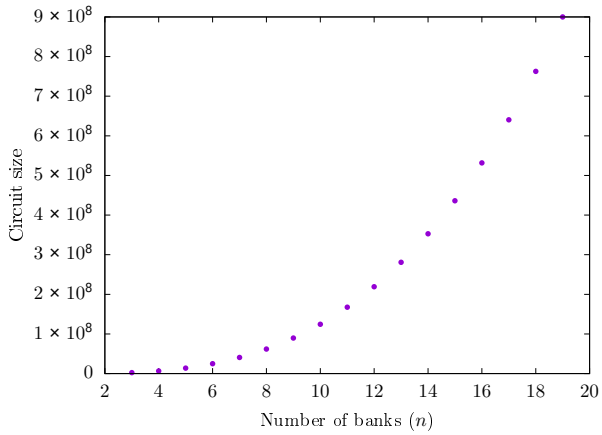


Figure 15: Circuit sizes for the market values algorithm (Algorithm 6) as implemented in EMP Wang et al. (2017). The plot shows the number of boolean gates required to securely execute the algorithm with $k = 10$.

In addition to *generating* circuits, EMP-toolkit can securely *execute* the circuits using circuit garbling Yao (1982, 1986), Bellare et al. (2012b,a), Beaver et al. (1990). EMP-toolkit offers three modes of secure computation, a 2-party protocol secure against semi-honest adversaries (sh2pc),

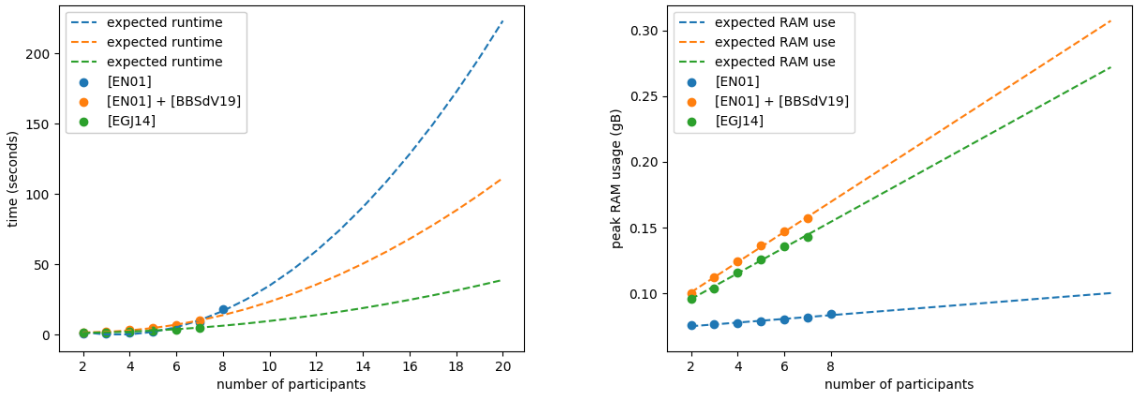
a 2-party protocol secure against malicious adversaries (ag2pc) and a multiparty protocol secure against malicious adversaries (agmpc).

We tested our circuits for correctness using the semi-honest 2-party protocol. Unfortunately, the multiparty EMP framework crashed with Segmentation Faults when we tried to securely execute the larger circuits securely, however, preliminary results are in Table 2.

Parties	Time (s)	RAM (gigabytes)
2	0.65	.15 -.19
4	14.58	1.97 - 3.92

Table 2: Resource requirements for EMP-toolkit to run Algorithm 6. In our experiments, one party required a larger amount of RAM, while the rest used only the smaller amount.

MPyC The results are in Figure 16. As expected, the iterative algorithm is much slower with larger numbers of parties. However, it also has a lower memory overhead.



(a) Wall-clock time.

(b) RAM usage.

Figure 16: Total resources required to run Algorithm 4 using the MPyC framework. We benchmarked our implementation for up to 15 parties on the BIS data described in Section 5.1.

A.4 Other financial network models

Our approach in this paper is general enough to be applied to other, more elaborate network models, including those that combine equity and debt. We provide an overview of some of these models below.

Allen and Gale (1998) considered a banking network where banks hold deposits and must choose whether to make short-term (liquid) or long-term (illiquid) investments. Their model has three

stages. At time $t = 0$, depositors deposit money and the deposit institutions apportion these deposits into short-term and long-term investments. At time $t = 1$, depositors may choose (probabilistically) to withdraw their deposits "early." If the deposit institution does not have enough liquid capital to cover these withdrawals, it must liquidate a long-term investment at a loss. At time $t = 2$, the remaining long-term investments yield returns to the bank. In follow-up work, Allen and Gale (2000), a network component was added, allowing banks to exchange deposits to mitigate risk, and these exchanges could lead to contagion effects.

Acemoglu et al. (2015) go beyond Allen and Gale (1998) to include interbank loans, and random "shocks." In their model, banks could choose to make short- or long-term investments, but long-term investments liquidated early (at time $t = 1$) would receive either a "low" return or a "normal" return. Banks that received the lower return were said to have received a "shock" and the crux of the work was to show how these shocks propagated through different types of extremal networks (e.g. the ring, and the complete network).

Morris (2000) developed a local contagion model, where each player plays a local game with each of its neighbors. Each player chooses a binary action for each game and receives payoffs according to a global 2 2 payoff matrix. Morris then analyzed how specific strategies spread through the network.

Eisenberg and Noe (2001) developed a simple network model where banks hold underlying assets (outside of the network), and are connected to each other via debt contracts. Eisenberg and Noe showed that under mild restrictions that in this model each bank has a unique "market value" at equilibrium and they gave a simple, iterative algorithm for computing the vector of market values. In Section 3 we go into more detail on the Eisenberg-Noe model and the algorithm for calculating market values. Gai and Kapadia (2010) considered an extension of the Eisenberg-Noe model, where banks could also hold illiquid assets, but we do not include that extension in our secure computations.

Elliott et al. (2014) considered a model like Eisenberg and Noe's, where banks hold underlying assets, but are connected via equity cross-holdings (e.g. shares) rather than fixed debts. In this model as well, under mild assumptions about the structure of the network, each bank has a unique market value at equilibrium. In § A we provide more detail about the Elliott-Golub-Jackson model.

Gouriéroux et al. (2012) proposed a model that combines liabilites à la Eisenberg and Noe (2001) and interbank investment à la Elliott et al. (2014). Similar models, combining both debt and equity have been further explored in Elsinger (2011), Jackson and Pernoud (2019).

B Technical Details

B.1 Secure Matrix Arithmetic

As discussed in §2.2, many secure computation systems are built around cryptographic secret-sharing, and the secure computation protocol operates by transforming secret shares of the input into secret shares of the output in a privacy-preserving way. See Beimel (2011) for a survey on cryptographic secret sharing.

Example 2. *Consider a financial network with 3 institutions, B_0, B_1, B_2. Institution i, knows its debts B_{ij} for $j \in \{0,1,2\}$, and its assets D_i. The institutions begin by writing B_{ij} in binary notation $(b_{i,j,0}, \ldots, b_{i,j,31}$, where $B_{ij} = \sum_{k=0}^{31} b_{ijk} 2^{k}$. Then each institution secret shares each bit b_{ijk}, by choosing two random bits, r_{ijk0} and $r_{ijk1} \in \{0,1\}$, and setting $r_{ijk2} = b_{ijk} + r_{ijk0} + r_{ijk1} \mod 2$. The private assets, D_i are cryptographically secret shared in the same manner.*

Once each participant has the secret shares, they can locally view these as secret shares of a liabilities matrix $\mathbf{L}$, and an asset vector $\vec{D}$. Then, the participants can use the oblivious market-clearing algorithm, Algorithm 4, to compute their market values based on their (secret) liabilities. When executing Algorithm 4, every addition (resp. multiplication) is replaced by the boolean circuit representing integer addition (resp. multiplication), and each gate of the circuit is executed securely on secret shares using a protocol like that described in Appendix B.1. Each comparison operation can be executed on bitwise secret-shares as described in Appendix B.2.1.

At the end of the algorithm, the participants all hold cryptographic secret shares of the market values $\vec{v}$. To reconstruct the actual market values, the participants send their shares of the relevant values to the relevant participant (e.g. sending shares of value i to institution i, or sending shares of all the values to the regulator).

To illustrate this type of system, we give a simple protocol for secure matrix multiplication, that is required for the algorithms we develop in §4 and §A.2.

B.1.1 Additive Secret Sharing

In this section, we review a basic additive secret sharing scheme for securely distributing and computing on private *matrices*. Let F be a finite field, and suppose player i has a secret matrix $\mathbf{X} \in \mathsf{F}^{k \times k}$. To secret share $\mathbf{X}$, player i will generate $n - 1$ uniformly random matrices $\mathbf{X}_1, \ldots, \mathbf{X}_{n-1}$ over $\mathsf{F}^{k \times k}$, and set

$$\mathbf{X}_n \stackrel{\text{def}}{=} \mathbf{X} - \sum_{j=1}^{n-1} \mathbf{X}_j$$

Then player i will send $\mathbf{X}_j$ to player j for $j \neq i$. The key features of the secret-sharing protocol are that

1. Privacy: Any collection of $n - 1$ shares $\{\mathbf{X}_j\}_{j \in S}$ with $S \subset [n]$ and $|S| < n$, is statistically independent of $\mathbf{X}$. This implies that if any collection of at most $n - 1$ players collude and reveal their shares to each other, they cannot learn *any* information about the matrix $\mathbf{X}$.

2. Linearity: If $\{\mathbf{X}_j\}$ is a secret sharing of $\mathbf{X}$, and $\{\mathbf{Y}_j\}$ is a secret sharing of $\mathbf{Y}$, with player j holding shares $\mathbf{X}_j$ and $\mathbf{Y}_j$, then player j can compute shares of the sum $\mathbf{X} + \mathbf{Y}$ by simply adding her shares $\mathbf{X}_j + \mathbf{Y}_j$. Note that this does not require any communication among the players (Algorithm 7). Similarly, if $\mathbf{X}$ is a private matrix, secret shared among the players, and $\mathbf{A}$ is a public matrix, known to all players, then each player can compute shares of the matrix $\mathbf{AX}$ by locally computing $\mathbf{AX}_i$ (Algorithm 8).

B.1.2 Matrix Computations on Shares

In this section, we outline protocols for performing linear-algebraic operations on matrices that are additively secret shared.

Throughout this section, we suppose $\mathbf{X}$ and $\mathbf{Y}$ are two private $k \times k$ matrices over a finite F, that are secret-shared, so that player i has matrices $\mathbf{X}_i$ and $\mathbf{Y}_i$ such that

$$\mathbf{X} = \sum_{i=1}^{n} \mathbf{X}_i$$
$$\mathbf{Y} = \sum_{i=1}^{n} \mathbf{Y}_i.$$

Algorithm 7 Addition of secret shares

Input: Player i holds secret shares $\mathbf{X}_i, \mathbf{Y}_i$
Player i computes $\mathbf{Z}_i = \mathbf{X}_i + \mathbf{Y}_i$
$\{\mathbf{Z}_j\}$ is a secret sharing of $\mathbf{X} + \mathbf{Y}$ ▷ $\mathbf{X} + \mathbf{Y} = \sum_j \mathbf{Z}_j$

Lemma 2. *Algorithm 7 is a secure computation protocol for computing* secret-shares *of the sum* $\mathbf{X} + \mathbf{Y}$ *in the sense of Definition 1.*

Lemma 3. *Algorithm 8 is a secure computation protocol for computing* secret-shares *of the product* $\mathbf{AX}$ *in the sense of Definition 1, where* $\mathbf{A}$ *is a public matrix, and* $\mathbf{X}$ *is a private matrix.*

Algorithm 8 Multiplication by a public matrix

Input: Player i holds secret shares $\mathbf{X}_i$
Input: Player i holds a common, public matrix $\mathbf{A}$
Player i computes $\mathbf{Z}_i = \mathbf{A}\mathbf{X}_i$
$\{\mathbf{Z}_j\}$ is a secret sharing of $\mathbf{AX}$ ▷ $\mathbf{AX} = \sum_j \mathbf{Z}_j$

Multiplication of two private matrices:

The linearity of the secret sharing scheme allow players to compute the sum of secret-shared matrices (Algorithm 7) and the product of a public matrix with a secret-shared matrix (Algorithm 8) using only local computations (*i.e.*, without any communication between the players).

Multiplying two secret-shared matrices requires communication between the participants, but can be achieved if the players hold correlated randomness (often called Beaver triples Beaver (1991)). We outline the multiplication protocol below. Suppose $\mathbf{X}, \mathbf{Y}$ are private, $n \quad n$ matrices, secret shared as $\{\mathbf{X}_i\}, \{\mathbf{Y}_i\}$.

We assume that in a *pre-processing* phase, players are dealt correlated randomness in the form of matrices $\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i$ such that $\mathbf{C} = \mathbf{AB}$, and

$$\mathbf{A} = \sum_j \mathbf{A}_j, \qquad \mathbf{B} = \sum_j \mathbf{B}_j, \qquad \mathbf{C} = \sum_j \mathbf{C}_j$$

Then, some basic algebra shows that

$$\begin{aligned} \mathbf{XY} &= (\mathbf{A}+\mathbf{X})(\mathbf{B}+\mathbf{Y}) \quad (\mathbf{A}+\mathbf{X})\mathbf{Y} \quad \mathbf{X}(\mathbf{B}+\mathbf{Y}) \quad \mathbf{AB} \\ &= (\mathbf{A}+\mathbf{X})(\mathbf{B}+\mathbf{Y}) \quad (\mathbf{A}+\mathbf{X})\mathbf{Y} \quad \mathbf{X}(\mathbf{B}+\mathbf{Y}) \quad \mathbf{C} \end{aligned}$$

The key observation is that $\mathbf{A}+\mathbf{X}$ is independent of $\mathbf{X}$, and $\mathbf{B}+\mathbf{Y}$ is independent of $\mathbf{Y}$, thus the matrices $\bar{\mathbf{A}} \stackrel{\text{def}}{=} \mathbf{A}+\mathbf{X}$ and $\bar{\mathbf{B}} \stackrel{\text{def}}{=} \mathbf{B}+\mathbf{Y}$ can be revealed publicly without compromising privacy.

Algorithm 9 Private matrix multiplication with pre-processing

Input: Player i holds secret shares $\mathbf{X}_i, \mathbf{Y}_i$
Input: Player i holds *correlation triples* $(\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i)$.
Players use the secure addition protocol (Algorithm 7) to compute $\bar{\mathbf{A}} \stackrel{\text{def}}{=} \mathbf{A}+\mathbf{X}$
Players reveal $\bar{\mathbf{A}}$ publicly
Players use the secure addition protocol (Algorithm 7) to compute $\bar{\mathbf{B}} \stackrel{\text{def}}{=} \mathbf{B}+\mathbf{Y}$
Players reveal $\bar{\mathbf{B}}$ publicly
Player 1 computes: $\mathbf{Z}_1 = \bar{\mathbf{A}}\bar{\mathbf{B}} \quad \bar{\mathbf{A}}\mathbf{Y}_1 \quad \mathbf{X}_1\bar{\mathbf{B}} \quad \mathbf{C}_1$
Player j computes: $\mathbf{Z}_j = \quad \bar{\mathbf{A}}\mathbf{Y}_j \quad \mathbf{X}_j\bar{\mathbf{B}} \quad \mathbf{C}_j$ ▷ $\sum_j \mathbf{Z}_j = \mathbf{XY}$

Lemma 4. *Algorithm 9 is a secure computation protocol for computing* secret-shares *of the product* $\mathbf{XY}$ *in the sense of Definition 1.*

Using Algorithm 9, the players can compute matrix products privately *if they have correlated randomness* in the form of these multiplication triples $\{\mathbf{A}_i\}, \{\mathbf{B}_i\}, \{\mathbf{C}_i\}$. Note that each multiplication triple $\{\mathbf{A}_i\}, \{\mathbf{B}_i\}, \{\mathbf{C}_i\}$ can only be used once, otherwise $\bar{\mathbf{A}} = \mathbf{X} + \mathbf{A}$ would leak information about $\mathbf{X}$. Thus the players need new shares of multiplication triples for every secure matrix product they are planning to compute.

In real-world MPC computations it is common for the players to generate (and consume) gigabytes of correlated randomness.

B.1.3 Matrix Inversion

Calculating the clearing vector in both the debt model (§4), and calculating market values in the equity model (Appendix, §A.2) require inverting matrix that is related to the private cross-holdings.

Most secure computation protocols require first expressing the computation as a *circuit* (e.g. Yao (1982, 1986), Goldreich et al. (1987), Ben-Or et al. (1988), Chaum et al. (1988)), but matrix inversion (unlike matrix multiplication) does not have a simple circuit representation. To see why this might be, consider implementing row-reduction using only the addition and multiplication operations (*i.e.*, without comparisons and branching).

Fortunately, in both cases we can approximate the matrix inversion using an iterated algorithm, which converges exponentially quickly to the inverse. These iterated algorithms are "MPC-friendly" and are outlined in §4 and in the Appendix, §A.2.

In recent work, however, special-purpose, secure matrix inversion routines have been developed and implemented in MPyC (Schoenmakers 2019). For comparison, we implement our algorithms in SCALE-MAMBA and EMP-Toolkit (using iterated approximations to matrix inversion) and MPyC (using their special-purpose matrix inversion protocol). See Section 5 for details on the implementations.

Note that the secure matrix inversion algorithm developed for MPyC only provides security when the participants are "semi-honest" and thus is incompatible with SCALE-MAMBA and EMP-Toolkit which provide security against *malicious* adversaries.

B.1.4 Generating Correlated Randomness

Algorithms 7, 8, 9 show that if the players have a store of "multiplication triples" they can securely perform matrix operations on secret-shared matrices. This reduces the problem of secure matrix

arithmetic to securely generating correlated randomness.

This creates a natural separation of the protocol into a pre-processing phase where the parties generate correlated randomness, and an online phase where the parties consume the randomness to perform the secure computation. Since the pre-processing phase does not depend on the parties' inputs, the correlated randomness can be generated in advanced and stored for later use. The online phase can then be extremely efficient as it does not require any cryptographic operations.

There are several methods for obtaining this correlated randomness, and most MPC protocols have a pre-processing phase, where the players generate multiplication triples that are later consumed in an "online" or function-dependent phase of the secure computation protocol.

Some MPC systems assume the existence of a "trusted dealer" whose only role is to generate correlated randomness and distribute it to the players. This method is extremely efficient since it does not require any cryptographic operations. Using this model, the trusted dealer never has access to any sensitive information, however, the dealer must be trusted to generate the shares uniformly, and not to collude with the players. (If the dealer gave more than $\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i$ to player i, this would allow player i to violate the privacy of the protocol.

In the absence of a trusted dealer, the players can use cryptographic tools to generate correlated randomness. The SPDZ protocol (Damgård et al. 2012) and its improvement, termed Overdrive (Keller et al. 2018) use somewhat homomorphic encryption to generate correlated randomness. The MASCOT protocol Keller et al. (2016) uses Oblivious Transfer (OT) Even et al. (1985).

B.1.5 An Alternative Algorithm for Matrix Inversion

In both the debt model (Equation 9) and the equity model (Equation 14), calculating the market value of each institution requires inverting a matrix. Traditional, schoolbook matrix inversion algorithms are not amenable to secure computation, because they have execution paths that are highly data-dependent (e.g. identifying pivot columns).

The matrices that arise when calculating clearing vectors are column sub-stochastic, so their inverses can be calculating by an iterative algorithm (as in Sections 4 and A.2).

An alternative method for secure matrix-inversion was proposed in Mohassel (2011) that essentially reduces the problem of secure matrix-inversion to insecure matrix inversion. We outline this method below.

Recall that in the debt model, the Algorithm 2 finds a solution to Equation 9 ($\vec{p} = \mathbf{A}\vec{p} + \vec{b}$). We can rewrite this as

$$\vec{p} = \begin{pmatrix} \mathbf{I} & \mathbf{A} \end{pmatrix} \begin{pmatrix} 1 \\ \vec{b} \end{pmatrix} \tag{16}$$

The circuit representation for matrix inversion is large, so simply performing the inversion under MPC will likely be extremely inefficient. The matrix inversion can be done efficiently, however, using MPC as a building block. As usual, we use the notation $[\![x]\!]$ to denote arithmetic secret shares of the value $x \in \mathbb{Z}/p\mathbb{Z}$. These algorithms can then be implemented using any secure multiparty computation protocol that can provide arithmetic operations (additions and multiplications) on these shares (e.g. Ben-Or et al. (1988)).

Algorithm 10 Securely inverting a matrix in $GL_d(\mathbb{Z}/p\mathbb{Z})$ Mohassel (2011)

Input $[\![A]\!]$, with $A \in GL_d(\mathbb{Z}/p\mathbb{Z})$.
for $i = 1, \ldots, m$ **do**
  Player i generates $R_i \in GL_d(\mathbb{Z}/p\mathbb{Z})$.
  Player i shares R_i as $[\![R_i]\!]$.
end for
Compute $[\![(\prod_{i=1}^{m} R_i)\, A]\!] = (\prod_{i=1}^{m} [\![R_i]\!])\, [\![A]\!]$.
Reveal $(\prod_{i=1}^{m} R_i)\, A$.
Compute $((\prod_{i=1}^{m} R_i)\, A)^{-1}$.
Compute $[\![A^{-1}]\!] = ((\prod_{i=1}^{m} R_i)\, A)^{-1} \prod_{i=m}^{1} [\![R_i]\!]$.
Return $[\![A^{-1}]\!]$.

Using this method, MPC is only used for matrix products. The above method requires computing $2n$ matrix products under MPC which can still be quite costly. We can reduce this complexity further if we are not worried about collusion. Instead of having *all* players generate a blinding matrix $\mathbf{R}_i$ it suffices to have only a single player i generate $\mathbf{R}_i$ and have a second player, j, obtain the matrix $\mathbf{R}_i$ $(\mathbf{I} \quad \mathbf{A})$ in the clear and invert it. As long as i and j do not collude this gives a secure method for solving the linear equation with only two matrix-multiplications under MPC. Clearly, we can adjust the number of blinding matrices from anywhere from 1 to n, which trades off collusion resistance for efficiency.

The problem with this approach is that secret sharing is done in some finite field $\mathbb{Z}/p\mathbb{Z}$, so the resulting inverse is the matrix inverse in $\mathbb{Z}/p\mathbb{Z}$ whereas most applications want the inverse over $\mathbb{R}$. To address this problem, Blom et al. (2019) developed an algorithm for securely computing $\mathrm{Adj}(\mathbf{A}) \in \mathbb{Z}^{d \quad d}$ and $\det(\mathbf{A}) \in \mathbb{Z}$ separately. By separating the numerator and denominator, both calculations only involve integers and thus can be embedded naturally in $\mathbb{Z}/p\mathbb{Z}$, and hence can be easily computed using secure computation protocols that operate natively in $\mathbb{Z}/p\mathbb{Z}$.

Algorithm 11 gives a simple procedure for sampling a random (secret-shared) invertible matrix along with its determinant. L_d^1 is the space of lower-triangular $d \quad d$ matrices with ones on the diagonal. U_d is the space of upper triangular $d \quad d$ matrices. Matrices from these spaces can be

sampled by sampling each of their coordinates uniformly and independently. For an upper-triangular matrix, the determinant is simply the product of its diagonal entries, and for matrices in L_d^1, the determinant is always 1.

Algorithm 12 uses Algorithm 11 as a building block to develop an extremely efficient protocol for secure matrix inversion.

Algorithm 11 Generating a sharing of a random matrix and its determinant Blom et al. (2019)[Protocol 2]

Input $d \in \mathbb{Z}^+$.
Generate $[\![L]\!]$ with $L \in_R L_d^1(\mathbb{Z}/p\mathbb{Z})$
Generate $[\![U]\!], [\![\det U)^{-1}]\!]$ with $U \in_R U_d(\mathbb{Z}/p\mathbb{Z})$
Compute $[\![R]\!] = [\![L]\!][\![U]\!]$.
Set $[\![\det(R)^{-1}]\!] = [\![\det(U)^{-1}]\!]$.
Return $[\![R]\!]$, $[\![\det(R)^{-1}]\!]$.

Algorithm 12 Securely inverting a matrix Blom et al. (2019)[Protocol 3]

Input $[\![A]\!]$ with $A \in \mathbb{Z}^{d \times d}$.
Use Algorithm 11 to generate $[\![R]\!], [\![\det(R)^{-1}]\!]$.
Compute $[\![RA]\!] = [\![R]\!][\![A]\!]$.
Reveal $[\![RA]\!]$.
Compute $X = (RA)^{-1}$.
Compute $[\![A^{-1}]\!] = X[\![R]\!]$.
Compute $[\![\det A]\!] = \det(RA)[\![\det(R)^{-1}]\!]$.
Compute $[\![\mathrm{Adj}(A)]\!] = [\![\det A]\!][\![A]\!]$.
Return $[\![\mathrm{Adj}\, A]\!]$,$[\![\det A]\!]$.

B.2 Data-oblivious algorithms

Consider a simple algorithm for computing $y = x^a$ for a positive integer a.

Algorithm 13 A data-*dependent* algorithm for computing x^a for $a \in \mathbb{Z}$.

$y = 1$
for $i = 1, \ldots, a$ **do**
  $y = y \cdot x$
end for
return y

Algorithm 13 is *not* data-oblivious, since the number of multiplications depends on the input a.

B.2.1 Obliviously Evaluating Conditionals

Obliviously algorithms cannot branch on input data, and thus special care needs to be taken when obliviously evaluating programs with conditionals ("if" statements). In most situations, conditionals are obliviously evaluated using a *multiplexer*. For example, consider the simple program

if $x = 0$ then
    $y = z$
else
    $y = w$
end if

This program can be rewritten without a conditional as

$$y = x \quad w + (1 \quad x) \quad z.$$

Comparisons can be handled similarly. If x and y are boolean values, then

$$x < y \ , \quad : x \wedge y.$$

Comparisons between integers can be handled bit-by-bit. For example, in EMP-toolkit Wang et al. (2016), the boolean circuit for comparing two 32-bit integers requires 158 boolean gates, and the comparison between two floats requires 449 boolean gates.

Some MPC compilers can automatically expand conditional statements into boolean circuits, whereas others require the programmer to do this by hand.

B.2.2 Gaussian Elimination

As discussed in Appendix B.1.5, matrix inversion is a particularly difficult task for secure computation because traditional matrix inversion algorithms (that rely on Gaussian Elimination) are highly data-dependent.

Algorithm 14 shows a standard pseudocode implementation of a row-reduction algorithm based on Gaussian Elimination. There are two key places where this algorithm exhibits a data-dependency. Line 3 shows that the number of iterations depends on the number of pivots found. This is not a serious obstacle, however, as we can always run the loop for n iterations. The major problem comes in Line 5 where the conditional statement depends on reading data from a location, i_{max},

that is data-dependent. Then, actually branching on the conditional leaks information because the sequence of operations are different in the two branches of the conditional. As discussed in Appendix B.2.1, the traditional solution for this kind of data-dependent branching is to evaluate both branches of the conditional and multiplex the results. Unfortunately, because of the structure of the for loop, these conditionals are nested, and thus evaluated all branches will require an *exponential* number of branches. Thus alternative techniques are needed.

Algorithm 14 Gaussian Elimination

```
1:  h = 1                                                          ▷ Initialize pivot row
2:  k = 1                                                          ▷ Initialize pivot column
3: while h  m and k  n do
4:     i_max = argmax_{i=h,...,m} (jA[i,k]j)                       ▷ Find the k-th pivot:
5:    if A[i_max, k] = 0 then                                      ▷ No pivot in this column, pass to next column
6:        k = k + 1
7:    else
8:        SwapRow(h, i_max)
9:        for i = h + 1, ..., m do                                 ▷ For all rows below pivot
10:           f = A[i,k]/A[h,k]
11:           A[i,k] = 0     ▷ Lower part of pivot column is 0  /* Do for all remaining elements in
    current row: */
12:           for j = k + 1, ..., n do
13:               A[i,j] := A[i,j]    A[h,j]    f
14:           end for
15:       end for
16:        h = h + 1                                                ▷ Increase pivot row
17:        k = k + 1                                                ▷ Increase pivot column
18:    end if
19: end whilereturn A
```

B.3 The Perron-Frobenius Theorem

Definition 3 (Spectral radius). *If* $\mathbf{A}$ *2* $\mathbf{C}^{n\ \ n}$ *is an* n n *matrix with eigenvalues* $\lambda_1, \ldots, \lambda_n$*, then the* spectral radius *of* $\mathbf{A}$ *is defined to be*

$$\rho(\mathbf{A}) \stackrel{def}{=} \max_i (j\lambda_1 j, \ldots, j\lambda_n j)$$

Definition 4 (Strong connectivity). *We say that a weighted, directed graph is* ϵ-strongly connected *if for any two vertices,* u, v*, there is a directed path from* u *to* v *such that the minimum edge weight along the path is at least* ϵ*.*

In other words, $G = (V, E)$ *is* ϵ*-strongly connected if for all* u, v *2* V*, there exists a set*

$f u_0, \ldots, u_t g$ V *such that*

- $u = u_0$
- $u_t = v$
- (u_i, u_{i+1}) 2 E *for* $i = 0, \ldots, t$ 1
- $\min_{i=0,\ldots,t\ 1} \mathrm{wt}((u_i, u_{i+1}))$ ϵ

Lemma 5. *If $G = (V, E)$ is an ϵ-strongly connected network, then for any partition of the vertices into two sets U, W, there exists an edge from U to W of weight at least ϵ,* i.e., *there exists u 2 U, and w 2 W, such that (u, w) 2 E, and* $\mathrm{wt}((u, w))$ ϵ.

Definition 5 (Primitive matrix). *A matrix $\mathbf{A}$ is called* primitive *if there exists a k 2* N *such that all entries of $\mathbf{A}^k$ are positive.*

Definition 6 (Irreducible matrix). *A matrix is called* irreducible *if for any i, j 2 $[n]$, there is a k (depending on i and j) such that (i, j)th coordinate of $\mathbf{A}^k$ is positive.*

Lemma 6. *If $\mathbf{A}$ is the adjacency matrix of a weighted, directed graph G, and G is* strongly connected, *then $\mathbf{A}$ is irreducible, and $\mathbf{I} + \mathbf{A}$ is positive.*

The Perron-Frobenius Theorem (Theorem 1) gives a useful characterization of the spectrum of irreducible matrices.

Theorem 1 (Sternberg (2010)[Theorem 9.1.1]). *Let $\mathbf{A}$ be an irreducible matrix*

1. *$\mathbf{A}$ has a positive (real) eigenvalue $\lambda_{\max}$ such that all other eigenvalues of $\mathbf{A}$ satisfy*

$$j\lambda j \quad \lambda_{\max}.$$

2. *$\lambda_{\max}$ has algebraic and geometric multiplicity one, and has an eigenvector $\vec{x}$ such that $\vec{x} > 0$,* i.e., *all coordinates of $\vec{x}$ are positive*
3. *Any non-negative eigenvector of $\mathbf{A}$ is a multiple of $\vec{x}$.*
4. *If $\vec{y}$ 0, and $\vec{y}$ 6= $\vec{0}$, and μ is a number such that $\mathbf{A}\vec{y}$ $\mu\vec{y}$, then $\vec{y} > 0$, and μ $\lambda_{\max}$, with $\mu = \lambda_{\max}$ if and only if $\vec{y}$ is a multiple of $\vec{x}$.*
5. *If* **0** **B** **A**, *and* **B** 6= **A**, *then every eigenvalue, σ of* **B** *satisfies $j\sigma j < \lambda_{\max}$.*

6. *All the diagonal minors* $\mathbf{A}_{(i)}$ *obtained by deleting the* i*th row and column of* $\mathbf{A}$ *have eigenvalues with absolute value strictly less than* $\lambda_{\max}$.

7. *If* $\mathbf{A}$ *is primitive, then all other eigenvalues of* $\mathbf{A}$ *satisfy* $|\lambda| < \lambda_{\max}$.

Corollary 1. *If* $\mathbf{A}$ *is a non-negative irreducible matrix with Perron-Frobenius eigenvalue* $\lambda_{\max} < 1$, *then*

$$\lim_{t \to \infty} A^t = \mathbf{0}$$

in addition $\mathbf{I} - \mathbf{A}$ *is invertible, and*

$$(\mathbf{I} - \mathbf{A})^{-1} = \sum_{k=0}^{\infty} \mathbf{A}^k.$$

B.4 Security Definitions

The security of cryptographic protocols is often defined using the notion of *indistinguishability* of random variables (Definition 7). Roughly, two families of random variables (indexed by natural numbers) are statistically (resp. computationally) indistinguishable, if the probability that any algorithm (resp. any polynomial-time algorithm) can successfully distinguish a single sample of one distribution from a single sample of the other decreases super-polynomially in the instance size.

Definition 7 (Indistinguishability). *Two families of random variables* $\{X_k\}_{k \in \mathbb{N}}$, $\{Y_k\}_{k \in \mathbb{N}}$ *are said to be* statistically indistinguishable, *denoted* $X_k \approx Y_k$, *if for all* $c > 0$, *there is an* K *such that for all* $k > K$,

$$\sum_z |\Pr[X_k = z] - \Pr[Y_k = z]| < k^{-c}. \tag{17}$$

Similarly, $\{X_k\}$ *and* $\{Y_k\}$ *are said to be* computationally indistinguishable *if for all probabilistic polynomial-time algorithms,* A, *and all* $c > 0$, *there is an* N *such that for all* $k > K$,

$$|\Pr[A(X_k) = 1] - \Pr[A(Y_k) = 1]| < k^{-c}. \tag{18}$$

Definition 1 defines security in terms of *indistinguishability*, *i.e.*, that a protocol is secure if the distribution of views engendered by running the protocol on different inputs are indistinguishable. An alternative definition of security – that of *simulation-based security* – is also common in the cryptographic literature. Roughly, a protocol achieves simulation-based security if the views of the participants can be efficiently *simulated* (*i.e.*, generated) from the output alone. Lemma 7 shows that if a protocol achieves this notion of simulation-based security, then it also achieves the notion

of indistinguishability-based securite (Definition 1).

Lemma 7 (Simulation-based security). *An n-party computation protocol for computing the function $f \stackrel{def}{=} (f_1, \ldots, f_n)$ is secure in the sense of Definition 1 if there exist a randomized algorithm S (called the "simulator") such that for any $T \quad [n]$ with $jTj = t$, $S\left(f f_i(\vec{x})g_{i2T}\right)$ outputs $\left\{\mathsf{view}_i^S\right\}_{i2T}$ such that the distributions*

$$\left\{f\mathsf{view}_i\left(\vec{x}\right)g_{i2T} \;\middle|\; f y_i g_{i2T}\right\} \quad \left\{\left\{\mathsf{view}_i^S\right\}_{i2T}\right\}. \tag{19}$$

C Proofs

Before proving Proposition 1, we begin by reviewing a basic property of *acyclic* networks.

Lemma 8. *If $\mathbf{A}$ is the adjacency matrix of a weighted, directed acyclic graph on n vertices, then $\mathbf{A}$ is nilpotent, and $\mathbf{A}^n = \mathbf{0}$.*

Proof. Since the graph is acyclic, there is an ordering of the vertices such that every edge connects a lower vertex to a higher one. In other words, there is an ordering of the vertices $v_1, \ldots, v_n$, such that if $e = (v_i, v_j)$ is an edge, then $i < j$. Under this ordering, the adjacency matrix is upper triangular with 0s on the diagonal. Thus there exists a permutation matrix $\mathbf{Q}$ such that

$$\mathbf{Q}\mathbf{A}\mathbf{Q}^T = \mathbf{U}$$

where $\mathbf{U}$ is an upper-triangular matrix with 0s on the diagonal. Then $\mathbf{U}$ is nilpotent, and $\mathbf{U}^n = \mathbf{0}$. Thus

$$\mathbf{A}^n = \mathbf{Q}^T\mathbf{U}^n\mathbf{Q} = \mathbf{0}.$$

□

Proof of Proposition 1. (i) First note that a solution, $\vec{x}$ to Equation $\vec{x} = \mathbf{A}\vec{x} + \vec{b}$ satisfies

$$(\mathbf{I} \quad \mathbf{A})\,\vec{x} \;= \vec{b}.$$

Corollary 1 shows that $\lim_{t\,!\,1} \mathbf{A}^t = \mathbf{0}$, thus $(\mathbf{I} \quad \mathbf{A})^{\,1}$ exists and

$$(\mathbf{I} \quad \mathbf{A})^{\,1} = \sum_{t=0}^{1} \mathbf{A}^t.$$

Now, let $\vec{x}$ denote a solution to $\vec{x} = \mathbf{A}\vec{x} + \vec{b}$, and let $\vec{e}^k$ denote the error at the kth step of

Algorithm 2, *i.e.*,

$$\vec{e}^{\,k} \stackrel{\text{def}}{=} \vec{x} \quad \vec{p}^{\,k}. \tag{20}$$

Then

$$\begin{aligned}
\vec{e}^{\,k} &= \vec{x} \quad \vec{p}^{\,k} \\
&= \mathbf{A}\vec{x} + \vec{b} \quad \left(\mathbf{A}\vec{p}^{\,k-1} + \vec{b}\right) \\
&= \mathbf{A}\left(\vec{x} \quad \vec{p}^{\,k-1}\right) \\
&= \mathbf{A}\vec{e}^{\,k-1}.
\end{aligned}$$

Thus

$$\vec{e}^{\,k} = \mathbf{A}^k\vec{e}^{\,0} = \mathbf{A}^k\left(\vec{x} \quad \vec{p}^{\,0}\right). \tag{21}$$

Thus by Corollary 1

$$\lim_{k \to \infty} \vec{e}^{\,k} = \lim_{k \to \infty} \mathbf{A}^k\vec{e}^{\,0} = \vec{0}. \tag{22}$$

Since $\mathbf{A} \neq \mathbf{0}$, if $\vec{p}^{\,0} \neq \vec{x}$, then $\vec{p}^{\,k} \neq \vec{x}$ for all $k > 0$.

(ii) Since the network is acyclic, Lemma 8 gives that $\mathbf{\Pi}^n = \mathbf{0}$. Thus if $\mathbf{A} \stackrel{\text{def}}{=} \Lambda\Pi^T\Lambda$ satisfies $\mathbf{A}^n = \mathbf{0}$. Then since $\vec{p}_{i+1} = \mathbf{A}\vec{p}_i + \vec{b}$,

$$\vec{p}_n = \mathbf{A}^n + \sum_{i=0}^{n-1} \mathbf{A}^i\vec{b} = \sum_{i=0}^{n-1} \mathbf{A}^i\vec{b},$$

and

$$\begin{aligned}
\vec{p}_{n+1} &= \mathbf{A}^{n+1} + \sum_{i=0}^{n} \mathbf{A}^i\vec{b} \\
&= \sum_{i=0}^{n-1} \mathbf{A}^i\vec{b} \\
&= \vec{p}_n
\end{aligned}$$

Thus $\vec{p}_n$ is a fixed point of Equation 9. □

Proposition 1 shows that when the network is *acyclic*, Algorithm 2 introduces no error, and thus the approximate market values calculated by Algorithm 4 are error-free. Most real-world networks, however, have cycles, but there are still situations when Algorithm 4 returns *exact* results.

Proposition 2 shows that if there are *no* failures in equilibrium, then Algorithm 4 returns the exact results (without error).

Proof of Proposition 2. (i) First, note that Algorithms 2 and 3 are data-oblivious, since the number of iterations in Algorithm 4 is fixed to be n, the overall algorithm is also data oblivious.

Note that at each iteration of Algorithm 4, if the defaulter list does not grow, then the defaulter list (and clearing vector) will remain constant throughout all subsequent iterations. Since the maximum number of defaulters is bounded by the network size, n, after n iterations the defaulter list will have stabilized, so (assuming Algorithms 2 and 3 were perfect), Algorithm 4 would compute the market valuations exactly.

To see that Algorithm 2 computes an approximate solution to Equation 9, first, notice that since the rows of Π sum to 1, we have the spectral radius of Π is 1, *i.e.*, $\rho(\Pi) = \rho\left(\Pi^T\right) = 1$.

Now, note that since the network is assumed to be strongly connected, the matrix Π^T is irreducible (See Lemma 6 in Appendix B.3), $\Lambda\Pi^T\Lambda \quad \Pi^T$, and thus we can apply the Perron-Frobenius Theorem (See Theorem 1 in Appendix B.3) to conclude that $\rho\left(\Lambda\Pi^T\Lambda\right) < 1$ whenever $\Lambda \neq \mathbf{I}$ (*i.e.*, whenever the defaulting set is not the complete set of institutions).

Thus by Proposition 1, Algorithm 2 (FindFix) converges exponentially quickly in $\rho(\Lambda\Pi^T\Lambda) < 1$. When $i = 1$, $\vec{p} = \bar{p}$, since $\bar{p}$ is an upper bound for the true market-clearing vector (for any defaulting set), Proposition 1 shows that $\vec{p}$ is greater than or equal to the true market-clearing vector throughout the entire course of Algorithm 4.

Next, we analyze the number of arithmetic operations needed to compute Algorithm 4. As explained in Appendix B.1, secure *linear* operations can be computed without communication between the participants, thus we focus on the number of *multiplications* and ignore the number of additions (which do not contribute to the communication costs of he protocol). Step 5 requires $2n$ scalar multiplications since Λ is a diagonal matrix. Step 6 requires $n^2 + 3n$ scalar multiplications. Step 7 requires kn^2 scalar multiplications. Step 8 requires n^2 multiplications and n comparisons. Step 12 requires n multiplications. Since each of these steps is repeated n times, the algorithm requires

$$(k+2)n^3 + 6n^2 \text{ multiplications}$$
$$n^2 \text{ comparisons}$$

(ii) Since there are no defaulters, the true market clearing vector is $\bar{p}$. In the first iteration of FindFix, $\Lambda = \mathbf{0}$, $\mathbf{A} = \mathbf{0}$, thus Equation 9 becomes $\vec{p} = \mathbf{O}\vec{p} + \bar{p}$, and $\bar{p}$ is clearly a solution. Since

no new banks fail with $\vec{p} = \bar{p}$, the clearing vector $\vec{p}$ remains fixed at $\bar{p}$ through every iteration, and both the true and approximate (oblivious) algorithm return $\bar{p}$. □

Proof of Lemma 1. Assume $\vec{v} \neq 0$ then since $C_{ij} \geq 0$, and $v_i \geq 0$, $\|\mathbf{C}\vec{v}\|_1 < (1 - s_{\min})\|\vec{v}\|_1$ Thus

$$\|(\mathbf{I} - \mathbf{C})\vec{v}\| \geq \|\mathbf{I}\vec{v}\| - \|\mathbf{C}\vec{v}\| > 0$$

so $\mathbf{I} - \mathbf{C}$ has a trivial kernel, and hence is invertible.

From here, the result follows just as in the scalar case: Let $\mathbf{S} = \sum_{k=0}^{N} \mathbf{C}^k$, then $\mathbf{S}(\mathbf{I} - \mathbf{C}) = \mathbf{I} - \mathbf{C}^{N+1}$. Thus

$$\sum_{k=0}^{N} \mathbf{C}^k = (\mathbf{I} - \mathbf{C})^{-1}(\mathbf{I} - \mathbf{C}^{N+1})$$

which means

$$(\mathbf{I} - \mathbf{C})^{-1} \sum_{k=0}^{N} \mathbf{C}^k = (\mathbf{I} - \mathbf{C})^{-1}\mathbf{C}^{N+1}$$

so

$$\left\| \left((\mathbf{I} - \mathbf{C})^{-1} \sum_{k=0}^{N} \mathbf{C}^k \right) \vec{v} \right\| = \left\| (\mathbf{I} - \mathbf{C})^{-1}\mathbf{C}^{N+1}\vec{v} \right\|_1 < \left\| (\mathbf{I} - \mathbf{C})^{-1}\vec{v} \right\|_1 (1 - s_{\min})^{N+1}$$

which gives the result.

□

Proof of Proposition 3. By Lemma 1, Line 5 returns an approximation $\mathbf{A}$ such that $\left\| \mathbf{A} - (\mathbf{I} - \mathbf{C})^{-1} \right\|_1 < (1 - s_{\min})^{k+1}$. Thus

$$\begin{aligned} \left\| \mathbf{S}\mathbf{A}\vec{D} - \mathbf{S}(\mathbf{I} - \mathbf{C})^{-1}\vec{D} \right\|_1 &= \left\| \mathbf{S}\left(\mathbf{A} - (\mathbf{I} - \mathbf{C})^{-1} \right)\vec{D} \right\|_1 \\ &\leq \left\| \left(\mathbf{A} - (\mathbf{I} - \mathbf{C})^{-1} \right)\vec{D} \right\|_1 \\ &\leq (1 - s_{\min})^{k+1} \left\| \vec{D} \right\|_1 \end{aligned}$$

The total computational cost of the algorithm can be computed as follows. Line 6 requires n^3 multiplications and n^3 additions. Line 9 requires $n^2 + n$ multiplications and n^2 additions. Thus the complete algorithm requires $kn^3 + n^2 + n$ multiplications and $kn^3 + n^2$ additions. □

Proof of Lemma 2. Security: In the protocol defined by Algorithm 7, there is *no* communication between the participants, so $\mathsf{view}_i(\mathbf{X}_i, \mathbf{Y}_i) = \emptyset$, which is trivially independent of the private inputs.

Correctness: At the end of the protocol, player i holds $\mathbf{Z}_i \stackrel{\text{def}}{=} \mathbf{X}_i + \mathbf{Y}_i$

$$\begin{aligned}\sum_j \mathbf{Z}_j &= \sum_j (\mathbf{X}_j + \mathbf{Y}_j) \\ &= \left(\sum_j \mathbf{X_j}\right) + \left(\sum_j \mathbf{Y}_j\right) \\ &= \mathbf{X} + \mathbf{Y}.\end{aligned}$$

□

Proof of Lemma 3. Security: In the protocol defined by Algorithm 8, there is *no* communication between the participants, so $\mathsf{view}_i(\mathbf{X}_i) = \text{;}$, which is trivially independent of the private inputs. Correctness: At the end of the protocol, player i holds $\mathbf{Z}_i \stackrel{\text{def}}{=} \mathbf{A}\mathbf{X}_i$

$$\begin{aligned}\sum_j \mathbf{Z}_j &= \sum_j (\mathbf{A}\mathbf{X}_j) \\ &= \mathbf{A}\left(\sum_j \mathbf{X_j}\right) \\ &= \mathbf{A}\mathbf{X}.\end{aligned}$$

□

Proof of Lemma 4. Security: In the protocol defined by Algorithm 9 $\mathsf{view}_i(\mathbf{X}_i, \mathbf{Y}_i) = \left\{\bar{\mathbf{A}}, \bar{\mathbf{B}}\right\}$. Now,

$$\begin{aligned}\bar{\mathbf{A}} &= \mathbf{A} + \sum_j \mathbf{X}_j \\ \bar{\mathbf{B}} &= \mathbf{B} + \sum_j \mathbf{Y}_j,\end{aligned}$$

since $\mathbf{A}$ and $\mathbf{B}$ are uniformly distributed in $\mathsf{F}^{k\ \ k}$, $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$ are uniformly distributed in $\mathsf{F}^{k\ \ k}$ and

are thus independent of $\mathbf{X}$ and $\mathbf{Y}$. Correctness: At the end of the protocol, player i holds $\mathbf{Z}_i$, and

$$\begin{aligned}
\sum_j \mathbf{Z}_j &= \mathbf{Z}_1 + \sum_{j=2}^{n} \mathbf{Z}_j \\
&= \bar{\mathbf{A}}\bar{\mathbf{B}} \quad \bar{\mathbf{A}}\mathbf{Y}_1 \quad \mathbf{X}_1\bar{\mathbf{B}} \quad \mathbf{C}_1 + \sum_{j=2}^{n} (\quad \mathbf{A}\mathbf{Y}_j \quad \mathbf{X}_j\mathbf{B} \quad \mathbf{C}_j) \\
&= \bar{\mathbf{A}}\bar{\mathbf{B}} \quad \mathbf{A}\left(\sum_{j=1}^{n} \mathbf{Y}_j\right) \quad \left(\sum_{j=1}^{n} \mathbf{X}_j\right)\mathbf{B} \quad \sum_{j=1}^{n} \mathbf{C}_j \\
&= (\mathbf{A}+\mathbf{X})(\mathbf{B}+\mathbf{Y}) \quad \bar{\mathbf{A}}\left(\sum_{j=1}^{n} \mathbf{Y}_j\right) \quad \left(\sum_{j=1}^{n} \mathbf{X}_j\right)\bar{\mathbf{B}} \quad \sum_{j=1}^{n} \mathbf{C}_j \\
&= (\mathbf{A}+\mathbf{X})(\mathbf{B}+\mathbf{Y}) \quad \bar{\mathbf{A}}\left(\sum_{j=1}^{n} \mathbf{Y}_j\right) \quad \left(\sum_{j=1}^{n} \mathbf{X}_j\right)\bar{\mathbf{B}} \quad \mathbf{AB} \\
&= (\mathbf{A}+\mathbf{X})(\mathbf{B}+\mathbf{Y}) \quad \bar{\mathbf{A}}\mathbf{Y} \quad \mathbf{X}\bar{\mathbf{B}} + \mathbf{AB} \\
&= (\mathbf{A}+\mathbf{X})(\mathbf{B}+\mathbf{Y}) \quad (\mathbf{A}+\mathbf{X})\mathbf{Y} \quad \mathbf{X}(\mathbf{B}+\mathbf{Y}) \quad \mathbf{AB} \\
&= \mathbf{XY}.
\end{aligned}$$

□

Proof of Lemma 5. Pick arbitrary vertices $u \in U$ and $w \in W$. By the strong connectivity of G, there exists a path $(u_0, \ldots, u_t)$ connecting u to w all of whose edges are of weight at least ϵ. Since $u \in U$ and $w \in W$, there must exist an index, i such that $u_i \in U$ and $u_{i+1} \in W$. Thus (u_i, u_{i+1}) is an edge connecting U to W, such that $\mathrm{wt}((u_i, u_{i+1}) \quad \epsilon$. □

Proof of Lemma 6. For an adjacency matrix $\mathbf{A}$, the (i,j)th entry of the matrix $\mathbf{A}^k$ gives the number of paths from i to j, thus $\mathbf{A}$ is irreducible if and only if there is some path from i to j for every $i, j \in [n]$.

If $\mathbf{A}$ is irreducible, then the binomial expansion

$$(\mathbf{I}+\mathbf{A})^k = \mathbf{I} + k\mathbf{A} + \frac{k(k \quad 1)}{2}\mathbf{A}^2 + \tag{23}$$

Will have all positive entries if k is large enough. □

Proof of Lemma 7. Suppose $\vec{x}$ and $\vec{x}'$ are two input vectors such that $f_i(\vec{x}) = f_i(\vec{x}')$ for $i \in T$.

Then

$$\left\{ f\mathsf{view}_i\left(\vec{x}\right)g_{i2T} \;\middle|\; fy_ig_{i2T} \right\} \quad S\left(ff_i(\vec{x})g_{i2T}\right) \quad S\left(\left\{f_i\left(\vec{x}^{\emptyset}\right)\right\}_{i2T}\right) \quad \left\{ \left\{\mathsf{view}_i\left(\vec{x}^{\emptyset}\right)\right\}_{i2T} \;\middle|\; fy_ig_{i2T} \right\}. \tag{24}$$

□

Proof of Corollary 1. Let $\vec{v}$ denote the eigenvector corresponding to eigenvalue λ_{max}. Then to prove the first claim, note that

$$\lim_{t \,!\, 1} \mathbf{A}^t\vec{v} = \lim_{t \,!\, 1} \lambda_{\mathsf{max}}^t\vec{v} = \vec{0}.$$

By the Perron-Frobenius Theorem (Theorem 1), $\vec{v}$ is positive, since $\mathbf{A}$ is non-negative, $\mathbf{A}^t$ is non-negative for all t, thus it must be that $\lim_{t \,!\, 1} \mathbf{A}^t = \mathbf{0}$.

For the second part, note that

$$(\mathbf{I} \quad \mathbf{A})\sum_{k=0}^{t}\mathbf{A}^k = \mathbf{I} \quad \mathbf{A}^{t+1}.$$

Taking limits as $t \;!\; 1$ gives

$$(\mathbf{I} \quad \mathbf{A})\sum_{k=0}^{1}\mathbf{A}^k = \mathbf{I},$$

thus

$$(\mathbf{I} \quad \mathbf{A})^{\;1} = \sum_{k=0}^{1}\mathbf{A}^k.$$

□