Scalable Secure Multi-Party Network Vulnerability Analysis via Symbolic Optimization

Kinan Dak Albab*, Rawane Issa†, Andrei Lapets‡, Azer Bestavros§, Nikolaj Volgushev¶

Department of Computer Science, Boston University
111 Cummington Mall, Boston, MA, 02215
*Email: *babman@bu.edu, †ra1issa@bu.edu, ‡lapets@bu.edu, §best@bu.edu, ¶nikolaj@bu.edu*

Abstract—**Threat propagation analysis is a valuable tool in improving the cyber resilience of enterprise networks. As these networks are interconnected and threats can propagate not only within but also across networks, a holistic view of the entire network can reveal threat propagation trajectories unobservable from within a single enterprise. However, companies are reluctant to share internal vulnerability measurement data as it is highly sensitive and (if leaked) possibly damaging. Secure Multi-Party Computation (MPC) addresses this concern. MPC is a cryptographic technique that allows distrusting parties to compute analytics over their joint data while protecting its confidentiality. In this work we apply MPC to threat propagation analysis on large, federated networks. To address the prohibitively high performance cost of general-purpose MPC we develop two novel applications of optimizations that can be leveraged to execute many relevant graph algorithms under MPC more efficiently: (1) dividing the computation into separate stages such that the first stage is executed privately by each party without MPC and the second stage is an MPC computation dealing with a much smaller shared network, and (2) optimizing the second stage by treating the execution of the analysis algorithm as a symbolic expression that can be optimized to reduce the number of costly operations and subsequently executed under MPC. We evaluate the scalability of this technique by analyzing the potential for threat propagation on examples of network graphs and propose several directions along which this work can be expanded.**

Index Terms—**Secure multi-party computation, Network distance, Symbolic expressions.**

1. Introduction

Risk assessment entails the evaluation of a metric that quantifies the separation between an entity of interest and other interconnected similar entities with known vulnerabilities. For example, in communications networks such as those operated by Internet Service Providers (ISPs), the closer a router is to a server that is likely to be the target of a distributed denial of service attack (DDoS), the higher the risk that this router will itself be victimized by DDoS traffic. Similarly, interconnected members of a coalition/consortium of firms [1] may consider the distance from each of their local network nodes to the closest "potentially compromised" nodes (e.g., running a yet-to-patch kernel) within a risk analysis of inter/intra-network threat propagation [2]. In social networks such as those defined by instant messaging, the closer a user is to accounts with a known history of illicit activities, the higher the risk that this user will be victimized. In financial contagion networks (e.g., those resulting from the use of derivatives) the closer a financial instrument is to a highly leveraged asset, the higher the risk of cascaded failures affecting that instrument.

In the above and many other examples, risk assessment boils down to proper modeling of an underlying network as a graph of (possibly weighted) vertices and edges, and to evaluation of network metrics using one of many well-known graph algorithms. Such metrics (e.g., shortest path, network flow, or betweenness centrality) are often network-wide, requiring *full knowledge* of the underlying graph to be computed accurately.

While seemingly straightforward, risk assessment is typically complicated by the fact that in relevant settings such as vulnerability assessment via traffic anomaly detection, the underlying network connecting entities to one another is composed of *privately managed subnetworks* (each under the control of an autonomous, self-interested party). Though every party is interested in – and capable of – measuring its own traffic to evaluate risks associated with its assets, *no single party has a full view of the entire underlying graph* that would allow it to evaluate the same risk assessment metrics *network-wide*. Moreover, the various parties that make up the system view the structure and self-initiated measurements of their private networks as highly-valued (and/or highly-sensitive) assets that constitute a significant investment and marketplace competitive advantage (and/or marketplace liability). As such, these parties may not be incentivized or willing to trust such assets (e.g., the "health" of their internal nodes or the topology of their internal networks) to any other party.

For individual parties, the circumstances described above may lead to lost opportunities to leverage valuable global information in conjunction with local measurements. Furthermore, investments that parties make in measuring their own networks may not maximize returns, and may even be redundant if they attempt to use partial information to approximate measurements of neighbor networks. Many of these issues could be addressed if it were possible to compute aggregate measurements on network data from multiple private networks without requiring sharing of sensitive information (this would also reduce risks and incentivize parties to work together to measure global network properties).

1.1. Secure Multi-Party Computation

In this paper, we tackle this problem by leveraging secure Multi-Party Computations (MPC) – a cryptographic construct that allows the computation of a function over a set of private(ly-held) inputs in such a way that the only information revealed by the computation is the output of the function. A key hurdle facing wide adoption of MPC is the computational inefficiencies associated with evaluating arbitrary functions over sizable inputs. Consequently, an important body of recent research on MPC has been concerned with the development of optimizations targeting specific algorithms or settings [3]. In this paper, we do so for a very important (and we believe very large) class of problems that involve the evaluation of "network distance" algorithms over an interconnection of private networks. To that end, we present our work towards a generalizable framework that consists of graph transformations and compile-time optimizations that could be integrated seamlessly into generic graph algorithm libraries, and eventually provided transparently to programmers through standard APIs.

1.2. Scenario and Approach

For ease of presentation and without loss of generality, we use the network resulting from interconnecting a set of private enterprise networks to illustrate the premise of our proposed framework. In particular, we consider the problem of assessing the risk of threat propagation in communication networks by computing the minimum distance (or hop count) between a private network node (a router or server in an enterprise network) and compromised nodes that may exist in other private networks. We assume that the set of private (enterprise) networks that make up the system are interconnected to one another via a set of *border gateways*. The border gateways and the topology connecting these border gateways to one another are assumed to be public, whereas the internal nodes in an enterprise network (as well as the topology connecting these nodes to one another) are assumed to be private (and thus not to be revealed).

One straightforward approach to using MPC to compute the distances between all pairs of nodes in the underlying network while preserving confidentiality of the topology and node state information of private networks is to convert the entire node distance algorithm into an MPC form using an existing technique or framework [4], [5]. However, performing operations under MPC can be prohibitively costly when compared to executing the same operations over the inputs "in the open". This added cost comes from the communication overhead and the increased complexity of analyzing data obliviously [3].

We present a method for optimizing (and thus making *feasible* for large-scale networks) the computation of node distances in a network under MPC (regardless of what particular MPC system is used). We assume that the overall network consists of many private sub-networks (each belonging to a distinct party) connected by edges between their public "gateway" nodes (each belonging to a sub-network). Using this method, each participating party only learns the distances between its own nodes and the closest "threat" nodes outside its sub-network.

We combine two techniques to improve the efficiency of the computation: separating the MPC computation into distinct local and global stages [6], [7] and using standard compiler optimization techniques [8] to reduce the number of costly operations in the MPC stage. These techniques can be applied to a variety of network analysis algorithms that can be run under any of a number of distinct MPC protocols.

The first technique is reducing the size of the input to the MPC computation. We split the computation into stages. The function to be computed (network distance) is applied locally by each party on their respective input. A portion of this stage's output (the gateway nodes) is then secret-shared between the parties and constitutes the input to the MPC stage. The gateway nodes need to be augmented with additional distance information to properly represent their local sub-networks in the secure computation. The results of the MPC stage are put back into the local sub-network, and the parties execute another round of local computation, yielding the distances from internal nodes.

The second technique is optimizing the MPC computation via symbolic optimization. The network distance problem is iterative in nature, and each iteration propagates the "threat" metric from each node to its neighbors. Nodes take the minimum threat from each neighbor. This requires computing a large number of minimum operations in every iteration, the results of which are fed into the next iteration. As minimum is expensive to compute in MPC, we use standard code and arithmetic optimization techniques [8], [9] to reduce the number of occurrences of the operator by (1) using loop unrolling to produce symbolic expressions describing the computation and (2) refactoring the expressions by exploiting the algebraic properties that govern the minimum and addition operators.

2. Problem Definition

Given an undirected graph G with a set V of nodes (nodes can be either safe or vulnerable), we must find the shortest distance between each node and the closest vulnerable node. We assume initially that vulnerable nodes have distance 0 and safe nodes have distance ∞. More formally, we are trying to compute $f(n) = \min_{v \in VN}(\mathsf{dist}(n, v))$, where VN is the set of vulnerable nodes and $\mathsf{dist}(n, v)$ is the shortest distance between nodes n and v.

Algorithm 1 computes network distance, where $\mathsf{nb}(n)$ is the set of neighbors of n and $D_i[n]$ is the distance computed

Algorithm 1 Network Distance

```
1: $i \leftarrow 0$
2: while $i <$ iterations do
3:     for $n \in V$ do
4:         $D_{i+1}[n] \leftarrow \min(D_i[n], \min_{v \in \mathsf{nb}(n)} (D_i[v] + 1))$
5:     $i \leftarrow i + 1$
```

at iteration i for node n. Notice that initially $D_0[n] = 0$ or ∞ if n was vulnerable or safe, respectively. The number of iterations can be set to the number of nodes. However, it is sufficient to set it to the diameter d of the graph. The runtime of the algorithm is $O(n \times d \times k)$, where n, d, k are the number of nodes, diameter, and degree, respectively. Notice that if n is fixed then d quickly decreases if k is increased (by Moore's upper bound [10]).

We consider input networks that are composed of two parts: (1) several private networks (each belonging to a party) for which the number of internal nodes and their interconnections are not public and each of which interacts with the rest through a relatively small number of publicly-known gateways; and (2) a public network that connects gateways of different parties. Note that this implies that the total number of gateways must be much smaller than the total number of nodes in the entire network. The number of gateway nodes per party and their connections with gateway nodes of other parties is assumed to be public information (note that an edge between two gateways of the same party is considered private and not part of the public network).

Security assumptions. The security model is assumed to be semi-honest [11]. Each party should learn only the distances at its own nodes, so the MPC stage yields different outputs to each party. The parties wish to hide the following.

1) The topology of a party's private network, including the number of nodes and their connections. This can be achieved at a significant performance cost when running MPC on the whole network by padding additional "fake" nodes. We achieve this in our three-round (local-MPC-local) approach without incurring additional costs.
2) The shortest path along which a vulnerable node propagates the "threat" to any of its gateway nodes.
3) A vulnerability's origin: from which node or which party's network a vulnerability has propagated.

Each party will know only the distances from each of its nodes to the closest threat. If the party is connected to multiple parties, or if the public network connecting the parties is sufficiently complex, there will be many paths along which the vulnerability could propagate, thus hiding the original source.

3. Computation Stages

Given our security assumptions, the computation cannot publicly refer to the internal topology of the parties' networks. Thus, the private networks (the number of nodes and their connections) and initial distances (0 or ∞) must be secret-shared as inputs to the computation. This adds a huge communication cost and complicates the computation, as it must operate on secret-shared network representations. Given current performance overheads of MPC, this approach is impractical for networks at realistic scales.

We propose doing the computation in multiple stages. First, each party will compute distances in its own private network (as if in isolation) to obtain the distances between all pairs of gateway nodes (i.e., the input to the MPC stage). This can greatly reduce the size of the input to the MPC stage (from the entire network to a much smaller sub-network). As the network connecting the gateways is public, we avoid the complications of secret sharing this network. We use Algorithm 1 for all stages of the computation.

Even with relaxed security assumptions that allow leaking information about the internal topology of the private networks, running MPC on the whole network will remain unrealistic due to the network size. In our approach, the MPC stage considers only the public network made out of the gateways and their connections, which is assumed to be much smaller than the entire network. Furthermore, the diameter of the public graph is known and can be used in the MPC stage to achieve a run-time of $O(n \times d \times k)$ as described in Section 2. The diameter of the entire network is not known (unless it is computed in MPC or the security assumption is relaxed further to reveal it). Thus, running MPC on the entire network requires using the number of nodes as an upper bound on the diameter, increasing running times to $O(n^3)$ for dense networks.

Local stage for inputs. In the first stage of the computation, each party performs the algorithm locally on its private network. We are only interested in computing the distances of the gateways. It may help to use a modified version of the algorithm that computes the distances for only the gateways without considering distant nodes that do not contribute.

A party's gateways may be connected internally inside its private network. These connection must not be ignored by the MPC stage since they can affect the outcome. For example, consider a case in which two parties networks are not connected directly but are connected to two different gateways of a third party (which are then connected internally by some path in that third party's local network). Failure to provide the MPC stage with information about the connection between these two gateways will cause vulnerabilities in the first party to be "invisible" to the second party and vice versa, preventing the vulnerability information from propagating.

Therefore, each party computes the shortest distance between every pair of gateway nodes. If two gateways are unconnected, the distance is set to be ∞. These distances will serve as "weights" to imaginary edges between gateways in the MPC stage. The actual value of the weights remains unknown as they are secret-shared. This will not reveal the existence (or nonexistence) of a path between two gateways, nor will it reveal its length. Thus, no information about the internal topology of any private network is leaked.

MPC stage. When all parties complete their local computations, they begin a multiparty computation. Each party secret shares (using Shamir secret sharing [12]) the distances of each gateway and the "weights" between every pair of gateways. Each party then builds two sets of expressions such that each gateway has a unique corresponding expression in each set. The first set is responsible for propagating distances over public edges between different sub-networks; the second set propagates distances over weighted imaginary edges connecting gateways in the same sub-network. These expressions are optimized and evaluated in MPC as explained in Section 4.

Local stage for outputs. Finally, each party receives the output distances of each gateway from the MPC stage, plugs the information into the result from the first local stage, and executes Algorithm 1 locally. This propagates distance information about local vulnerabilities as well as external vulnerabilities learned from the output of the MPC stage. The distances of the gateway nodes remain unchanged and the final distances of the internal nodes are computed.

4. Expression Trees and Symbolic Evaluation

Algorithm 2 MPC Stage

```
1:  $E1_0 = \{n \to X_n \mid n \in \mathsf{gateways}\}$        ▷ Public edges.
2:  for $i \leftarrow 1$ to $d$ do        ▷ $d$: Public graph diameter.
3:      for $n \in \mathsf{gateways}$ do
4:          $m = \min_{v \in \mathsf{nb}(n)} (E1_{i-1}[v] + 1)$
5:          $E_i[n] = \mathsf{simplify}(\min(E1_{i-1}[n], m))$
6:  $E2 = \{\}$        ▷ Weighted edge expr.'s.
7:  for $n \in \mathsf{gateways}$ do ▷ $\mathsf{gnb}(n)$: gateways in party of $n$.
8:      $m = \min_{v \in \mathsf{gnb}(n)} (X_v + \mathsf{weight}(v, n))$
9:      $E2[n] = \min(X_n, m)$
10: $valuation_0 = \{X_n \leftarrow \mathsf{secretshare}(n) \mid n \in \mathsf{gateways}\}$
11: for $i \leftarrow 1$ to $|\mathsf{parties}| - 1$ do
12:     $tmp = \{\}$
13:     for $n \in \mathsf{gateways}$ do
14:         $tmp[X_n] = \mathsf{eval}(E1_d[n], valuation_{i-1})$
15:     for $n \in \mathsf{gateways}$ do
16:         $valuation_i[X_n] = \mathsf{eval}(E2[n], tmp)$
17: for $n \in \mathsf{gateways}$ do
18:     $result[X_n] = \mathsf{eval}(E1_d[n], valuation_{|\mathsf{parties}|-1})$
```

We propose to derive a symbolic expression that describes an execution of the algorithm on a particular input graph. Each expression is an unfolding of algorithm iterations via symbolic execution. This yields two kinds of expressions (described below); every gateway will have a single expression of each type. Minimum operations are computationally expensive in MPC since they include several comparisons and multiplications on secret shares (which have a high communication overhead). Therefore, we optimize these expressions by reducing them to a single minimum operation with fewer arguments.

Public edge expressions. The first kind of expression is responsible for propagating distance information over the edges connecting different sub-networks. These are built by symbolically executing Algorithm 1 with symbolic variables as initial values. The abstract syntax for these expressions is defined as follows, where X_n is the variable representing the gateway n, and z is an integer representing a distance between gateways belonging to different sub-networks:

$$e \quad ::= \quad z \mid X_n \mid e + \ldots + e \mid \mathbf{min}(e, \ldots, e)$$

Furthermore, the particular algorithm being used ensures that every expression tree consists of alternating levels of **min** and addition operators. A **min** operator represents a single iteration for a single node in the execution of the algorithm, and its children are addition expressions that add the expressions from the previous iteration to either weights or to the integer constant 1 (line 4 in Algorithm 1).

Our goal is to reduce the number of **min** operators while pushing them up towards the root of the expression tree by exploiting the algebraic properties of $+$ and **min**. This is accomplished using three kinds of reductions.

In a *Plus-Min* reduction, an expression of the form $\mathbf{min}(e_0, ..., e_n) + e'_0 + ... + e'_m$ is reduced by moving the addition operations inside the **min** operator to produce $\mathbf{min}(e_0 + \overline{e}, ..., e_n + \overline{e})$ where $\overline{e} = e'_0 + ... + e'_m$. The resulting **min** operation has the same number of arguments as the original; the e'_i are guaranteed to not contain a **min** operator since the algorithm only adds 1.

In a *Min-Min* reduction, expressions of the form $\mathbf{min}(\mathbf{min}(e_0, ..., e_n), ..., \mathbf{min}(e'_0, ..., e'_m))$ are flattened to a single operation $\mathbf{min}(e_0, ..., e_n, e'_0, ..., e'_m)$ via associativity.

In an *Early-Min* reduction, any duplicates or arguments that are provably larger than at least one other argument are removed in expressions of the form $\mathbf{min}(e_0, ..., e_n)$ (where the e_i do not contain any **min** operators). This ensures that each gateway's variable appears as an argument at most once, so the maximum number of arguments in each expression matches the number of gateways.

We alternate between the first two reductions starting from the bottom of the expression tree and moving upwards (as the tree itself alternates between addition and **min** operators), yielding an equivalent single **min** operation. We apply the third reduction to the resulting tree. Applying the reductions in every iteration to keep expression sizes small can speed up the simplification (lines 1–5 of Algorithm 2).

Weighted edge expressions. The gateways of each party can be connected by a path of internal nodes; if such paths are not included in the computation we may get incorrect results. Each party will therefore compute the shortest distances between every pair of its gateways; this second kind of expression is responsible for propagating the distance information through these paths. The abstract syntax for these expressions is defined below ($W_{n,v}$ represents the weight between gateways n and v, where $W_{n,v} \equiv W_{v,n}$):

$$e \quad ::= \quad X_n \mid W_{n,v} \mid X_n + W_{n,v} \mid \mathbf{min}(e, \ldots, e)$$

An addition expression containing X_n may only contain one weight between n and some other gateway v belonging to the same party. These expressions are constructed by lines 6–9 of Algorithm 2 and do not need to be simplified. Each gateway n has a single expression of this kind of the form $\mathbf{min}(X_n, X_{v1}+W_{n,v1}, X_{v2}+W_{n,v2}, \ldots)$; the argument count is equal to the number of gateways in the party.

Expression evaluation. The symbolic expressions are evaluated using a valuation map that assigns secret shares of values to variables. This map differs by party since each has different shares of each value. When all expressions are evaluated, parties send their shares of every gateway value to the party of that gateway. Parties combine the shares of their gateways to learn the distances at these gateways without learning the distances at others' gateways.

Iterative evaluation of the expressions is required. The first kind of expression propagates distances coming from gateways connected by a path of public edges; the second kind propagates distances through the weighted edges to allow distances connected by a combination of public edges and a single weighted edge to propagate. In the worst case, two gateways may have as many weighted edges separating them as there are parties (e.g., two gateways at opposite ends of a chain of sub-networks). Thus, we must repeatedly evaluate the weighted expressions as many times as there are parties while evaluating the public edge expressions in between. The result of each evaluation is the new valuation of the respective gateway. This is implemented in lines 10–18 of Algorithm 2. At the end of the MPC stage, we only evaluate public edge expressions (the subsequent local stage achieves the effect of evaluating weighted edge expressions).

The symbolic execution and simplification process does not include any MPC constructs or message passing and can be executed quickly. Expressions themselves are evaluated under MPC. There are two expressions per gateway, each containing no more arguments than the number of gateways. Thus, the total size of all expressions is $2 \times g \times g$. Since we repeat the evaluation for each party, the total run-time in MPC for p parties and g gateways is $O(p \times g^2)$. This mirrors the run-time for Algorithm 1 with the diameter set to p and the degree set to g (since every gateway is connected by a weighted edge to every other gateway in its party).

5. Tool-set and Implementation

We implemented a Python library, ExpressionMPC [13], that allows programmers to quickly write code segments that execute symbolically and evaluate in MPC. It provides classes (simplifiers) that implement the reductions from Section 4 and an evaluator that uses VIFF [4] to secret share inputs between parties (as well as to interpret the expressions using generic VIFF operators). The results are opened and shared only with appropriate parties. Code written using this library need not include explicit MPC constructs or parse-tree manipulations, but expert users can explicitly include reduction operations to improve performance. The library provides overloaded operators for building expression trees, and users of the library can write their own simplifiers or evaluators and use them in combination with those provided.

We ran our implementation on networks representing autonomous systems peering information [14]. Each party's sub-network was one of these networks in its entirety. Gateways were selected randomly from each sub-network and connected by edges drawn from a scale-free distribution. We used one Amazon EC2 c4.large instance per party (2.9GHz, 2 cores, 3.75GB RAM). The benchmarks in Table 1 show that the dominating factor is the number of gateways per party, which agrees with our analysis (as the number of weights is quadratic in the number of gateways per party).

TABLE 1. EXPERIMENTAL RESULTS

P	Node	Edge	Gateway	Pub. Edg.	Our Method	MPC[1]	Clear
3	32378	67218	34	86	0.72min	$>$ 24hrs	1.7s
3	32378	67218	220	579	62min	$>$ 24hrs	2s
4	43510	89783	43	105	2.75min	$>$ 24hrs	2.8s
4	43510	89783	301	850	72min	$>$ 24hrs	2.8s
5	55093	156773	45	105	2min	Rec. Limit[2]	5.8s
5	55093	156773	393	981	154min	Rec. Limit[2]	5.9s
10	108788	250800	44	124	3min	-	19.4s

6. Related Work

Prominent theoretical approaches to MPC [3] are based on linear secret sharing (LSS) [15] and garbled circuits [16]. Recent efforts to deliver these approaches to programmers yielded many publicly-available implementations [4], [17], [5], [18], [19], [20], [6], [21]. The past few years have also seen successful deployments of MPC in production [22], [23]. Sepia [17] is an LSS-based framework offering protocols for semi-honest majority that are optimized to reduce latency in order to support near real-time data processing. The authors of Sepia propose aggregating traffic volume information across ISPs to detect systemic traffic anamolies–the supported metrics are heuristics such as top-k queries; our work complements this effort by enabling more sophisticated analytics such as network flow. VIFF [4] implements an asynchronous protocol for LSS-based MPC over arithmetic circuits with suites for dishonest minority and actively-malicious adversaries. VIFF was chosen as our prototype's backend due to its convenient integration with Python, support for a variable number of parties and threat models, and code base quality. While most MPC frameworks operate by translating a program into an arithmetic or boolean circuit, alternatives exist [24], [25]. For example, there exists an iterative approach to computing shortest distance metrics under MPC [25] in which iteratively leaking and using information that can be inferred from the final output throughout the protocol execution improves performance significantly. However, this technique does not directly apply to our scenario because it relies on all computed distance values being revealed to all participating parties; this entails far more leakage than revealing gateway node distances to their respective owners. Investigating how

1. The direct implementation of Algorithm 1 in MPC on the entire graph. This keeps the initial distances and origin of vulnerabilities private but reveals all the nodes and connections in all the sub-networks. The security guarantees of this approach are a strict subset of our security guarantees.

2. Python maximum recursion depth was reached after 20 hours. The recursion depth in the computation was 10000 (the default is 1000).

such optimizations can be tailored to match our requirements is promising future work. Our work builds on earlier efforts to explore how decomposing MPC protocols into stages enables new optimizations and trade-offs [7], [26].

7. Conclusion and Future Work

We have described – and demonstrated the relative performance advantages of – novel techniques that enable scalable application of MPC to the calculation of network-wide metrics in federated networks. We hope to apply these techniques to other algorithms, eventually assembling a general-purpose framework for defining network analyses that scale well when executed under MPC. We believe our approach of dividing computations into stages to reduce input sizes and symbolically optimizing expressions that represent executions can apply to a variety of algorithms for which a publicly available termination condition exists. In particular, we are investigating how this technique can be used when computing minimum spanning trees (e.g., using Borůvka's algorithm) and maximum flows. We also plan to extend the library to support more expression patterns, operators, and algebraic laws beyond the existing support for the associative and distributive properties of addition and minimum. In certain algorithms, it should be possible to transform loops and conditionals into expressions in a similar manner. Finally, we plan to implement and test the same algorithms using other MPC frameworks, both to obtain a performance baseline and to evaluate scalability on real-world large-scale network data sets.

Acknowledgments: This work benefited from discussions and feedback from Mayank Varia and Ran Canetti, and was partially supported by National Science Foundation Grants #1414119 and #1430145.

References

[1] "ACSC: MassInsight Advanced Cyber Security Center," http://www.acscenter.org/, [Accessed: March 9, 2017].

[2] K. M. Carter, N. C. Idika, and W. W. Streilein, "Probabilistic threat propagation for network security," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 9, pp. 1394–1405, 2014.

[3] D. W. Archer, D. Bogdanov, B. Pinkas, and P. Pullonen, "Maturity and performance of programmable secure computation," *IEEE Security Privacy*, vol. 14, no. 5, pp. 48–56, Sept 2016.

[4] "VIFF, the Virtual Ideal Functionality Framework," http://viff.dk/, [Accessed: March 9, 2017].

[5] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A Framework for Fast Privacy-Preserving Computations," in *Proceedings of the 13th European Symposium on Research in Computer Security*, ser. Lecture Notes in Computer Science, S. Jajodia and J. Lopez, Eds., vol. 5283. Springer Berlin / Heidelberg, 2008, pp. 192–206.

[6] A. Rastogi, M. A. Hammer, and M. Hicks, "Wysteria: A programming language for generic, mixed-mode multiparty computations," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, ser. SP '14, 2014, pp. 655–670.

[7] N. Volgushev, M. Schwarzkopf, A. Lapets, M. Varia, and A. Bestavros, "DEMO: Integrating MPC in Big Data Workflows," in *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security*, Vienna, Austria, October 2016.

[8] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools (2Nd Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006.

[9] R. Kastner, A. Hosangadi, and F. Fallah, *Arithmetic Optimization Techniques for Hardware and Software Design*, 1st ed. New York, NY, USA: Cambridge University Press, 2010.

[10] A. J. Hoffman and R. R. Singleton, "On moore graphs with diameters 2 and 3," *IBM J. Res. Dev.*, vol. 4, no. 5, pp. 497–504, Nov. 1960.

[11] O. Goldreich, *Foundations of Cryptography: Volume 2, Basic Applications*. New York, NY, USA: Cambridge University Press, 2004.

[12] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[13] "ExpressionMPC," https://github.com/hicsail/ExpressionMPC, [Accessed: March 9, 2017].

[14] "Stanford large network dataset collection: Autonomous systems (as) peering information inferred from oregon route-views," https://snap.stanford.edu/data/, [Accessed: March 9, 2017].

[15] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, ser. STOC '88, 1988, pp. 1–10.

[16] A. C. Yao, "Protocols for secure computations," in *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, ser. SFCS '82, 1982, pp. 160–164.

[17] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos, "Sepia: Privacy-preserving aggregation of multi-domain network events and statistics," in *Proceedings of the 19th USENIX Conference on Security*, ser. USENIX Security'10, 2010, pp. 15–15.

[18] M. Keller, P. Scholl, and N. P. Smart, "An architecture for practical actively secure mpc with dishonest majority," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security*, ser. CCS '13, 2013, pp. 549–560.

[19] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi, "Oblivm: A programming framework for secure computation," in *IEEE S & P*, 2015.

[20] K. Nayak, X. S. Wang, S. Ioannidis, U. Weinsberg, N. Taft, and E. Shi, "Graphsc: Parallel secure computation made easy," in *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. IEEE Computer Society, 2015, pp. 377–394.

[21] D. Malkhi, N. Nisan, B. Pinkas, Y. Sella *et al.*, "Fairplay-secure two-party computation system." in *USENIX Security Symposium*, vol. 4. San Diego, CA, USA, 2004.

[22] I. Damgård, K. Damgård, K. Nielsen, P. S. Nordholt, and T. Toft, "Confidential benchmarking based on multiparty computation," Cryptology ePrint Archive, Report 2015/1006, 2015, http://eprint.iacr.org/.

[23] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach, and T. Toft, "Financial cryptography and data security," R. Dingledine and P. Golle, Eds., 2009, ch. Secure Multiparty Computation Goes Live, pp. 325–343.

[24] A. Shelat and M. Venkitasubramaniam, "Secure computation from millionaire," in *Proceedings of the 21st International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2015, pp. 736–757.

[25] J. Brickell and V. Shmatikov, "Privacy-preserving graph algorithms in the semi-honest model," in *Proceedings of the 11th International Conference on Theory and Application of Cryptology and Information Security*. Springer-Verlag, 2005, pp. 236–252.

[26] N. Volgushev, A. Lapets, and A. Bestavros, "Programming Support for an Integrated Multi-Party Computation and MapReduce Infrastructure," in *Proceedings of the Third IEEE Workshop on Hot Topics in Web Systems and Technologies*, Washington, D.C., USA, November 2015.