Web-based Multi-Party Computation with Application to Anonymous Aggregate Compensation Analytics

Andrei Lapets Eric Dunton Kyle Holzinger Frederick Jansen Azer Bestavros

CS Dept., Boston University
111 Cummington Mall
Boston, MA USA 02215
{lapets, edunton, kholz, fjansen, best}@bu.edu

Abstract

We describe the definition, design, implementation, and deployment of a multi-party computation protocol and supporting web-based infrastructure. The protocol and infrastructure constitute a software application that allows groups of cooperating parties, such as companies or other organizations, to collect aggregate data for statistical analysis without revealing the data of individual participants. The application was developed specifically to support a Boston Women's Workforce Council (BWWC) study of the gender wage gap among employers within the Greater Boston Area. The application was deployed successfully to collect aggregate statistical data pertaining to compensation levels across genders and demographics at a number of participating organizations.

1 Introduction

Modern organizations, including companies, educational institutions, and governments agencies, have been collecting and analyzing data pertaining to their internal operations for some time and to great effect, such as in evaluating performance or improving efficiency. While this data is of great value to the organizations themselves, it is likely that novel insights valuable to multiple organizations, to policymakers, or to society at large can be derived by combining data from these multiple organizations and analyzing it as a single corpus.

Unfortunately, the data collected by organizations internally is often proprietary and confidential, and its release may be potentially deleterious to their interests. Furthermore, while organizations may have the option of releasing sensitive data selectively to specific agents entrusted with its analysis, this presents a security risk: how will the data be physically transferred in a secure way, how will it be housed during the analysis, and how will it be destroyed after an analysis is complete?

Secure multi-party computation (MPC) techniques have been known for decades at least as theoretical constructs [25], and recent efforts [19, 13, 16, 21, 23] are finally bringing us closer to a point at which these techniques will be available to end-users (i.e., organizations interested in collectively analyzing their sensitive data).

In this report, we describe the definition, design, implementation, and deployment of a multi-party computation protocol and supporting web-based infrastructure for analyzing compensation data (broken down by gender and demographics) from a collection of employer organizations. The secure multi-party computation protocol utilized for this application is of relatively modest

mathematical sophistication: it allows multiple parties to compute a sum total of a collection of secret quantities. Each participating party's secret quantity is known only to that party and is never revealed to any other individual party. Nevertheless, the sum is computed and can be made available to all participants.

The definition of the protocol and the design and implementation of the software were influenced heavily by the practical restrictions and challenges of the particular target application. Only limited resources were available to develop the software itself, both in terms of time and funding. Furthermore, the software was designed to be deployed easily and rapidly (i.e., requiring no setup, no specialized hardware, and no specialized software) in order to minimize any hurdles that might discourage participation. Finally, the software was designed to be used within a relatively narrow time window by non-experts whose technical expertise did not need to go far beyond familiarity with spreadsheet software and web browsing clients. All of these constraints presented difficulties that are not usually addressed by protocol definitions, or even by existing MPC frameworks that target users who have at least some technical expertise.

2 Protocol Definition

The protocol developed for this application is a variant of a technique that allows multiple parties to collectively compute a sum of their own individual quantities without revealing those quantities to one another [17].

2.1 Drawbacks and Security Definition

There is a practical drawback to the "secure sum" protocol in its original form: the participating parties must all coordinate with one another to pass data in sequence from one party to another so that it visits each party exactly once. From an implementation standpoint, this would require a relatively sophisticated software infrastructure involving multiple client applications that would all communicate with one another and maintain state throughout the duration of the computation. Assuming limited software development resource, this complexity could make the software implementation process itself more difficult and the application more error-prone. The synchronization requirement would also make it more difficult to schedule and coordinate participation from a large number of parties, as each party would need to run the application for the duration of the computation (which may span hours or days). Finally, the sequenced approach does not support idempotent updates from participants; if even one participant makes an error and wishes to resubmit their data, the entire protocol would need to be restarted.

There are other practical drawbacks to the original protocol that were not addressed by our variant. In particular, collusion between two or more parties may reveal the secret data being aggregated during the execution of the protocol. Also, a party may misbehave by submitting data that is inaccurate or that corrupts the sum in a way that makes it difficult to derive meaning from it (e.g., by submitting very large values that skew an aggregate metric such as an average). Thus, the secure and correct operation of our protocol variant assumes that all participating parties have an incentive not to deviate from the protocol and have secured their own private data on their own premises.

The particular form of security that is guaranteed by our protocol variant is that any malicious outside party that can observe and permanently store *all* the communications between any and all participants will gain no information beyond the aggregate being computed. We exploit this feature of the protocol in the implementation: the server used for housing the data communicated between parties does not need to be secure and can be commodity hardware purchased from any

third-party provider. The security of our protocol relies on RSA [22] or any equivalent public-key cryptographic protocol.

2.2 Protocol Definition

The protocol requires the participation of three kinds of parties:

- the *aggregator* stores the data and enables communication between the participants (there is only one aggregator and it need not be secure or trusted);
- the *trusted party* is computing the actual aggregate sum (there is only one trusted party, and it must not collude with the aggregator or share any non-aggregate data with any other party);
- a *participant* is one of the parties contributing private data to the aggregate computation (the number of participants is finite and fixed for a single execution of the protocol, but is otherwise not limited).

A single execution of the protocol, which we will call a *session*, requires some preparation by the trusted party. The aggregator is fully automated, and the participants only need to submit their data at least once at any point after the initial preparation step is completed by the trusted party and before the final aggregate is computed by the trusted party. A protocol execution proceeds as follows:

(1) the trusted party initiates the process by generating a private and public RSA key pair and a unique session identifier, submitting the public key p to the aggregator, and sending the session identifier $j \in \mathbb{N}$ to all the participants (the last step is only for ease of data organization if there are multiple sessions[1]);

(2) we assume each of the n participants already possesses a rational number $d_i \in \mathbb{Q}$ where i is the index of the participant and $1 \leq i \leq n$ (this is the participant's contribution to the aggregate sum computation), and that each participant performs the following steps at least once[2]:

   (a) generate a private random rational number $m_i \in \mathbb{Q}$ which will act as the *random mask*,

   (b) calculate $r_i = d_i + m_i$ which will constitute the *masked data*,

   (c) submit r_i unencrypted to the aggregator by sending it over the untrusted network,

   (d) retrieve p to encrypt m_i to obtain c_i so that only the trusted party can read it, and

   (e) submit c_i to the aggregator by sending it over the untrusted network;

(3) the aggregator computes the sum of the masked data values $R = \sum_{i=1}^{n} r_i$ to obtain the aggregate masked data quantity R;

(4) the trusted party then performs the final steps of the protocol:

   (a) retrieve R and all the $c_1, \ldots, c_n$ from the aggregator over the untrusted network,

[1] The session identifier can serve another purpose beyond data organization: as long as no malicious agent possesses the session identifier, any data submitted by malicious agents will be ignored during the computation of the aggregate at the end of the protocol.

[2] Each participant can perform step (2) as many times as they wish before step (3) occurs; the operation they perform is idempotent if they always submit the same data.

(b) decrypt the encrypted random masks c_i using the private key to obtain the random masks m_i and compute the sum of the random masks $M = \sum_{i=1}^{n} m_i$ to obtain the aggregate mask M;

(c) compute the difference $D = R - M$ to obtain the true sum of the original unmasked data $D = \sum_{i=1}^{n} d_i$, which can then be shared with any interested parties.

Note that the correctness of the protocol is ensured by the commutativity and associativity of addition:

$$\begin{aligned} D &= R - M \\ &= \sum_{i=1}^{n} r_i - \sum_{i=1}^{n} m_i \\ &= \sum_{i=1}^{n} (d_i + m_i) - \sum_{i=1}^{n} m_i \\ &= \sum_{i=1}^{n} d_i + \sum_{i=1}^{n} m_i - \sum_{i=1}^{n} m_i \\ &= \sum_{i=1}^{n} d_i \end{aligned}$$

Note that the aggregator never sees the true random masks because they are always encrypted using the trusted party's public key, and the trusted party never sees the individual masked data entries unless it violates its promise not to collude with the aggregator (one may choose to have the aggregator keep a running total and immediately discard every masked data entry submitted to it, but this is not a theoretical guarantee and merely makes it more difficult but not impossible for a malicious trusted party to obtain that data).

The diagram in Figure 1 illustrates an example deployment of the protocol for two participants.

2.3 Example

To make the protocol more concrete, we consider an example with two participants: Alice and Bob. We suppose that Alice possesses the secret quantity $d_{\text{Alice}} = 34$ and Bob possess the secret quantity $d_{\text{Bob}} = 12$. Alice and Bob can proceed as follows, each performing step (2):

(a) Alice generates a random mask $m_{\text{Alice}} = 65$ and Bob generates a random mask $m_{\text{Bob}} = 70$;

(b–c) Alice and Bob compute their masked data values

$$\begin{aligned} r_{\text{Alice}} &= d_{\text{Alice}} + m_{\text{Alice}} \\ &= 34 + 65 \\ &= 99 \\ r_{\text{Bob}} &= d_{\text{Bob}} + m_{\text{Bob}} \\ &= 12 + 70 \\ &= 82 \end{aligned}$$

and submit them to the aggregator;

(d–e) Alice and Bob encrypt 65 and 70 to submit c_{Alice} and c_{Bob} to the aggregator, respectively.

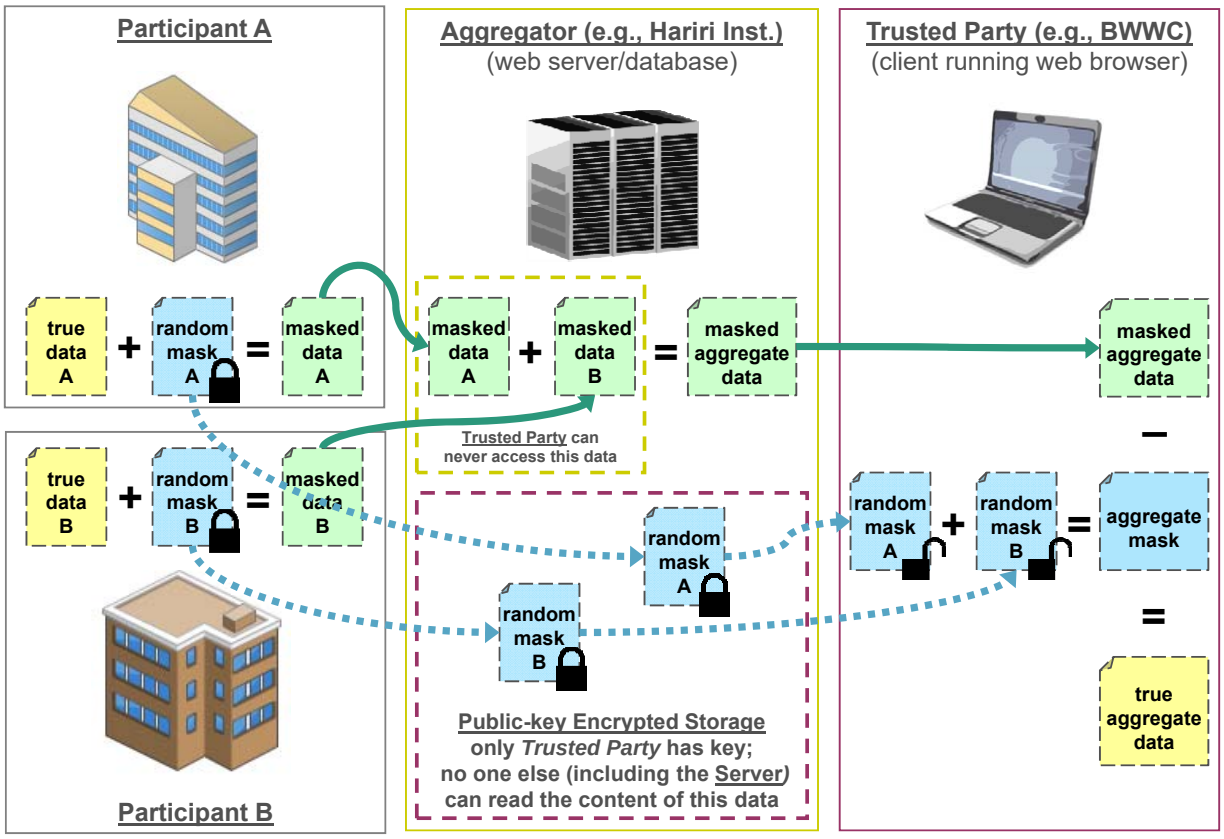

Figure 1: Illustration of a deployment of the protocol implementation for two participants.

The aggregator then computes $R = r_{\text{Alice}} + r_{\text{Bob}} = 99 + 82 = 181$ and stores this quantity. At this point, the trusted party can retrieve and decrypt c_{Alice} and c_{Bob} to obtain $m_{\text{Alice}} = 65$ and $m_{\text{Bob}} = 70$, and then it can also retrieve $R = 181$. The trusted party can then compute the sum:

$$\begin{aligned} d_{\text{Alice}} + d_{\text{Bob}} &= (r_{\text{Alice}} - m_{\text{Alice}}) + (r_{\text{Bob}} - m_{\text{Bob}}) \\ &= (r_{\text{Alice}} + r_{\text{Bob}}) - (m_{\text{Alice}} + m_{\text{Bob}}) \\ &= R - (m_{\text{Alice}} + m_{\text{Bob}}) \\ &= 181 - (65 + 70) \\ &= 181 - 135 \\ &= 46 \end{aligned}$$

Since $34 + 12 = 46$, the trusted party has computed the sum. However, the trusted party has not seen the individual values r_{Alice} and r_{Bob}, so it could not possibly recover the original data quantities. Likewise, the aggregator has only seen the encrypted masks c_{Alice} and c_{Bob}, so it could not derive the true data quantities, either.

3 Client Interface and Back-end Server Implementations

The purpose of the software application is to allow a group of non-expert participants to execute a session of the protocol defined in Section 2. In particular, the software application automates

all portions of the protocol except the initiation of a session (which can be done with a single manual click), the distribution of the session identifier (which are simply delivered to participants via email), and the entry of participant data (participants must use the client-facing interface to paste or enter the data before submitting). Since realistic scenarios involve not one numeric quantity per participant but a collection or table of labelled quantities, the software application actually implements the protocol in parallel on multiple labelled fields within a table. Beyond these features, the software application was developed under the constraints already enumerated in Section 2.1. These constraints informed the design and implementation decisions for the software application.

The two main components of the application are (1) the back-end server that acts as the aggregator and as the delivery mechanism for the client-facing interfaces, and (2) the client-facing interactive interfaces to be used by the trusted party and participants. The server is implemented using the Node.js [9, 26] framework and server-side data is stored within an instance of MongoDB [6, 24]. The Node.js application interacts with that instance using the Mongoose module [7]. The client-facing interface is implemented using JavaScript, employing the jQuery [4], Underscore.js [12], Handsontable [3], and JSEncrypt [5] libraries, in particular.

As both the server and client applications are authored using JavaScript, critical components such as the aggregate computation routines are shared between the components, reducing the likelihood of errors and facilitating maintenance and updates to the application. The actual fields of the table displayed to users are programmatically generated and can be modified. The JSEncrypt library is used to employ 1024 bit RSA encryption for encrypting the participant masks. The participant masks themselves are generated as arrays of pseudo-random numbers using the `window.crypto` object [14], which uses a high amount of entropy from the host operating system to ensure that any patterns that could be discovered in the generated numbers are minimized. Of course, it is worth noting that if a participant uses a compromised implementation of a web browser, the implementation of the `window.crypto` object may not conform to the published specification.

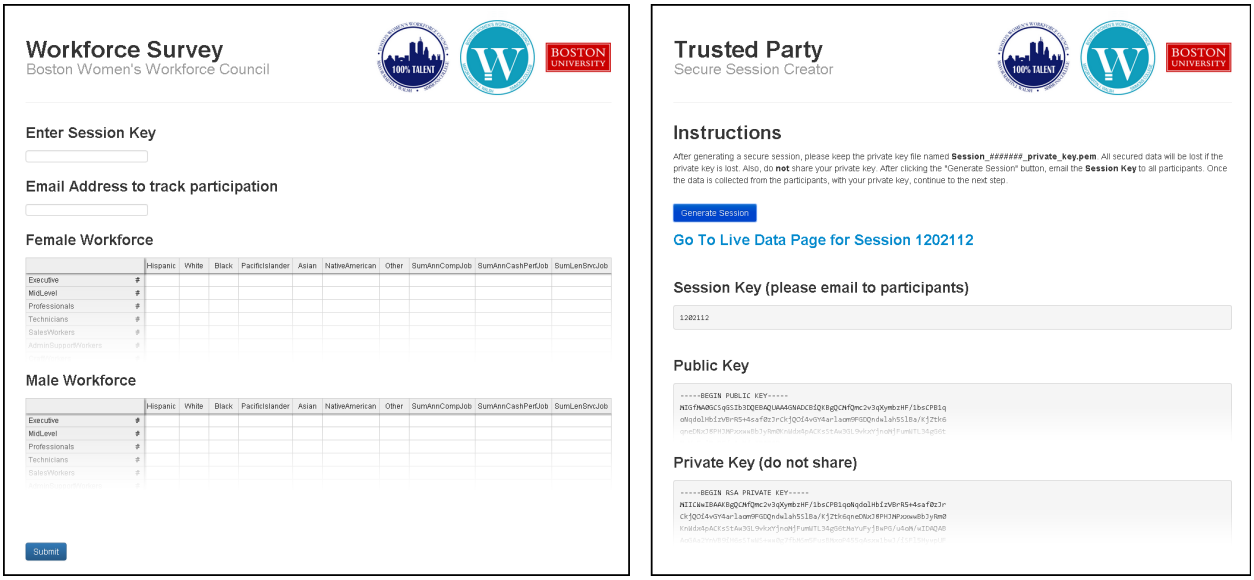

Figure 2: The participant (left) and trusted party (right) interfaces as they appear within a web browser.

Figure 2 illustrates two of the primary web interfaces as they appear within a web browser to the respective parties. The participant interface provides a familiar spreadsheet table that an

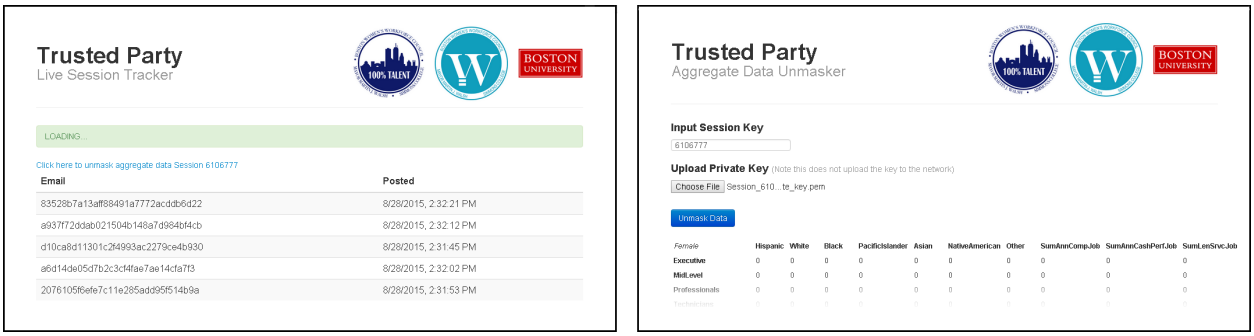

Figure 3: The session tracking (left) and data unmasking (right) interfaces as they appear within a web browser to the trusted party during a session and at the end of a session, respectively.

end-user can fill with data either manually or by pasting the data from another application. The email address is hashed on the client side and this hashed value is used only as an index into the server database, allowing each participant to submit more than once in a session (overwriting their previous submissions). The trusted party interface allows the trusted party to start a session, obtaining a session identifier and saving a private RSA key to their local storage. There are also two other interfaces for the trusted party, shown in Figure 3: a session tracker that displays how many participants submitted data, and a final unmasking and computation interface that requires them to supply the private key to the local client interface running in their browser in order to compute the final aggregate data. If too few participants have submitted their data, the application will not allow the trusted party to compute the final aggregate data. Once the final aggregate data is computed, it is displayed in the same familiar table format as the input table presented to individual participants.

As we mentioned above in Section 2.1, the security of the server housing the database and delivering the client interfaces over the web to user browsers is not required to ensure the security of the overall protocol. Thus, all the interfaces are publicly accessible.

4 Conclusions and Future Work

The protocol and software application described in this work were deployed successfully in Spring of 2015 to collect compensation data for the purpose of pay-equity analytics [20, 15]. Most practical issues deploying the software application involved browser version incompatibility, human error in entering data, and the duration and scheduling of the session. The simplicity of the protocol and its implementation was crucial in helping decision makers feel confident that they understood its operation and the security guarantees and contingencies that participating entailed. At the conclusion of the session, a group of about 40 employers was able to anonymously and securely contribute sensitive data without having to share or release the data itself.

A more general-purpose platform enabling this type of privacy-preserving collective analysis has far reaching potential for public initiative research studies. Such a platform can allow for larger, more in-depth analyses and could accommodate other research needs where sensitive information from multiple parties must be collectively processed in order to identify trends, diagnose problems, or test hypotheses. Areas of application range from smart cities [10], to genomics [8], to cybersecurity [18, 11].

Deploying any such platform involves overcoming not only technical and logistical challenges, but also challenges that may be appropriate to address under the umbrella of social science, orga-

nizational psychology, behavioral economics, or similar areas of study. One particular experience we had deploying the application was that despite the guarantees provided by the protocol, some participants still required that the trusted party and the operator of the aggregator sign a non-disclosure agreement governing the individual data submitted by those participants, even though that data contained no meaningful information content.

5 Acknowledgements

We would like to acknowledge all the members of the Boston Women's Workforce Council (BWWC), and to thank in particular Christina M. Knowles and Katie A. Johnston, who led the effort to organize participants and deploy the protocol as part of the 100% Talent: The Boston Women's Compact effort [1, 2]. We would also like to acknowledge the Boston University Initiative on Cities, and in particular Executive Director Katherine Lusk, who brought this potential application of secure multi-party computation to our attention. Both the BWWC and the Initiative on Cities contributed funding to complete this work. We would also like to acknowledge the Hariri Institute at Boston University for contributing research and software development resources. Support was also provided in part by Smart-city Cloud-based Open Platform and Ecosystem (SCOPE), an NSF Division of Industrial Innovation and Partnerships PFI:BIC project under award #1430145, and by Modular Approach to Cloud Security (MACS), an NSF CISE CNS SaTC Frontier project under award #1414119. Finally, we must thank the many representatives from the participant organizations for their interest, diligence, and feedback throughout the orientation, testing, and deployment process.

References

[1] 100% Talent: The Boston Women's Compact. `http://www.cityofboston.gov/women/workforce/compact.asp`. [Accessed: August 15, 2015].

[2] Boston: Closing the Wage Gap. `http://www.cityofboston.gov/images_documents/Boston_Closing%20the%20Wage%20Gap_Interventions%20Report_tcm3-41353.pdf`. [Accessed: August 15, 2015].

[3] Handsontable. `http://handsontable.com/`. [Accessed: August 15, 2015].

[4] jQuery. `https://jquery.com/`. [Accessed: August 15, 2015].

[5] JSEncrypt. `http://travistidwell.com/jsencrypt/`. [Accessed: August 15, 2015].

[6] MongoDB. `http://mongodb.org/`. [Accessed: August 15, 2015].

[7] Mongoose. `http://http://mongoosejs.com//`. [Accessed: August 15, 2015].

[8] NCI Cancer Genomics Cloud Pilots. `https://cbiit.nci.nih.gov/ncip/nci-cancer-genomics-cloud-pilots/`. [Accessed: August 15, 2015].

[9] Node.js. `http://nodejs.org/`. [Accessed: 15-August-2015].

[10] SCOPE: A Smart-city Cloud-based Open Platform and Ecosystem. `https://www.bu.edu/hic/research/scope/`. [Accessed: August 15, 2015].

[11] ThreatExchange. `https://threatexchange.fb.com/`. [Accessed: August 15, 2015].

[12] Underscore.js. http://underscorejs.org/. [Accessed: August 15, 2015].

[13] VIFF, the Virtual Ideal Functionality Framework. http://viff.dk/. [Accessed: August 15, 2015].

[14] Web Cryptography API. https://dvcs.w3.org/hg/webcrypto-api/raw-file/tip/spec/Overview.html. [Accessed: August 15, 2015].

[15] Rich Barlow. Computational Thinking Breaks a Logjam. http://www.bu.edu/today/2015/computational-thinking-breaks-a-logjam/. [Accessed: August 15, 2015].

[16] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A Framework for Fast Privacy-Preserving Computations. In Sushil Jajodia and Javier Lopez, editors, *Proceedings of the 13th European Symposium on Research in Computer Security - ESORICS'08*, volume 5283 of *Lecture Notes in Computer Science*, pages 192–206. Springer Berlin / Heidelberg, 2008.

[17] Chris Clifton, Murat Kantarcioglu, Jaideep Vaidya, Xiaodong Lin, and Michael Y. Zhu. Tools for privacy preserving distributed data mining. *SIGKDD Explor. Newsl.*, 4(2):28–34, December 2002.

[18] Lucian Constantin. IBM opens up its threat data as part of new security intelligence sharing platform. http://www.infoworld.com/article/2911154/security/ibm-opens-up-its-threat-data-as-part-of-new-security-intelligence-sharing-platform.html. [Accessed: August 15, 2015].

[19] Yehuda Lindell and Benny Pinkas. Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality*, 1(1):5, 2009.

[20] Joanne Lipman. Let's Expose the Gender Pay Gap. http://www.nytimes.com/2015/08/13/opinion/lets-expose-the-gender-pay-gap.html. [Accessed: August 15, 2015].

[21] Chang Liu, Xiao Shaun Wang, Kartik Nayak, Yan Huang, and Elaine Shi. Oblivm: A programming framework for secure computation. In *IEEE S & P*, 2015.

[22] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

[23] Kartik Nayak, Xiao Shaun Wang, Stratis Ioannidis, Udi Weinsberg, Nina Taft, and Elaine 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*, pages 377–394. IEEE Computer Society, 2015.

[24] Eelco Plugge, Tim Hawkins, and Peter Membrey. *The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing*. Apress, Berkely, CA, USA, 1st edition, 2010.

[25] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[26] Lambert M. Surhone, Mariam T. Tennoe, and Susan F. Henssonow. *Node.Js*. Betascript Publishing, Mauritius, 2010.