Privacy-Preserving Privacy Profile Proposal Protocol

Wyatt Howe
Boston University
whowe@bu.edu

Andrei Lapets
Nth Party, Ltd.
andrei@nthparty.com

Abstract

Many web-based and mobile applications and services allow users to indicate their preferences regarding whether and how their personal information can be used or reused by the application itself, by the service provider, and/or by third parties. The number of possible configurations that constitute a user's preference profile can be overwhelming to a typical user. This report describes a practical, privacy-preserving technique for reducing the burden users face when specifying their preferences by offering users data-driven recommendations for fully-specified preference profiles based on their inputs for just a few settings. The feasibility of the approach is demonstrated by a browser-based prototype application that relies on secure multi-party computation and uses the web-compatible JIFF library as the backbone for managing communications between the client application and the recommendation service. The principal algorithms used for generating proposed preference profiles are k-means clustering (for privacy-preserving analysis of preference profile data across multiple users) and k-nearest neighbors (for selecting a proposed preference profile to recommend to the user).

1 Introduction

It is common practice (and increasingly a regulatory requirement) to include within web-based and mobile applications facilities that allow users to express their preferences regarding whether and how their information will be collected, stored, processed, or shared with third parties. This flexibility leads to an overwhelmingly large space of configurations that users may find difficult to navigate. We propose a solution that allows users to specify their preferences for only a small subset of an overall profile. This small subset is then used in conjunction with a data set of preference profiles collected from other users to assemble a complete preference profile. This complete profile can then be proposed to the user for their approval or further modification.

The design of our proposed protocol and its implementation as a web-based application is informed by the following criteria.

- Choosing a privacy profile should be simple.
- No one other than the original user (not even the service) should have access to a user's preference profile data.
- Users should be able to receive helpful suggested profiles after specifying only a few initial preferences.

Adhering to these criteria, the implementation of our protocol relies on privacy-preserving secure computation techniques to classify a user's preferences based on the similarity of their partial profile to those of other users and to generate a proposal for the remaining portion of their profile. Our approach is able to satisfy the data privacy criterion by leveraging secure multi-party computation

(MPC) [16, 19], which enables workflows and web services that operate on encrypted data without decrypting it. Use of MPC makes it possible to offer the benefits of workflows and services to users while reducing or eliminating the need to store their sensitive data [5, 12].

2 Background and Related Work

There is a growing awareness among consumers of how organizations protect users' personal data, and new regulations such as the General Data Protection Regulation in the European Union [7] and the California Consumer Privacy Act [1] in the United States are being introduced. These factors are creating a marketplace for technological solutions that can allow organizations to continue offering their services while (1) being more transparent with users about how their data is stored and used, (2) giving users control over how their data can be used and shared with third parties, (3) leveraging traditional cybersecurity techniques such as instituting access controls to enforce data access policies within workflows involving multiple partner organizations, and (4) minimizing or eliminating storage and use of user data in a decrypted form by employing secure computation techniques such as multi-party computation.

The usability of cryptographic tools and techniques has been an active area of study for at least two decades [15, 18]. In parallel, there is a growing body of work on ways to measure user perceptions of how (and how well) systems protect user privacy [8] and ways to design and manage privacy features in complex user-facing applications and services [20]. The motivation for our work partially overlaps with the goals of these ongoing efforts: to offer users features such as control over their data while mitigating the burden of actually utilizing these features.

Another aspect of our work relates to the privacy-preserving properties both (1) of the web service or application itself *and* (2) of the underlying infrastructure and algorithms that are used to enhance usability. Proposed solutions for the first issue fall on a spectrum that ranges from dedicated runtime environments that can enforce data access policies [17] to secure computation techniques that can scale across contemporary web service and mobile application infrastructures [6]. In relying on secure multi-party computation, our work falls closer to the latter side of this spectrum. With regard to the second issue, a common approach to measuring and enhancing the usability of services and applications is to collect data about how users behave and then to use that data to drive improvements or to aid users automatically [13]. This approach may seem to be in conflict with the goal of improving the privacy-preserving characteristics of those same services and applications by limiting the extent to which user data can be utilized and exposed. However, secure computation resolves this conflict by allowing organizations to collect such data for evaluation purposes [14] and to use such data within usability-enhancing features (as demonstrated by the work described in this report).

3 Protocol Design and Definition

The protocol involves three stages in the flow of data: (1) submission of user preferences profiles, (2) processing of user preference profiles, and (3) derivation and sharing of recommended profiles with users. We define an individual *preference* as a single integer representing one of several choices, and define a *preference profile* as a collection of such choices. There are two types of parties in this design: (1) the recommendation servers, and (2) the users who submit preferences and receive recommendations. All servers symmetrically employ the same MPC algorithm, with one server designated as being responsible for coordination and for returning user recommendations.

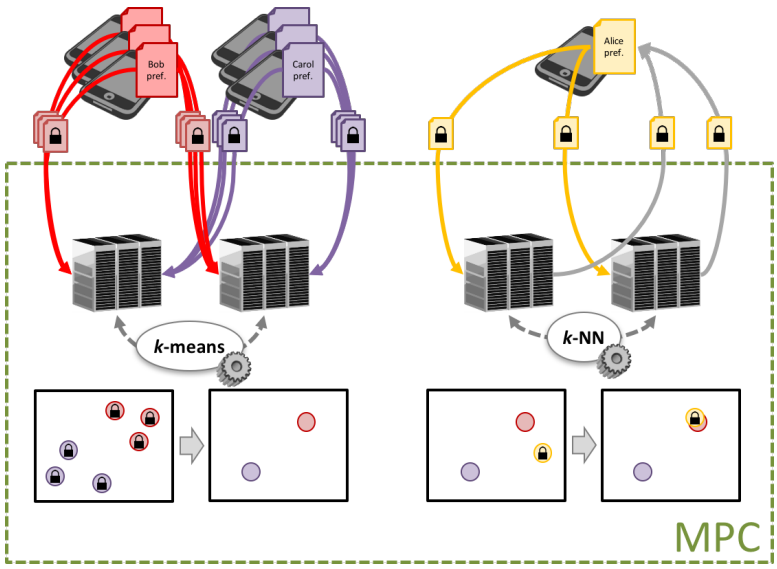

Figure 1: Architecture and workflow diagram.

3.1 Preference Profile Submission Stage

After a user builds a partial profile, the app submits that partial profile to the recommendation servers. To protect the privacy of the user's chosen preferences, the protocol uses Shamir's Secret Sharing scheme [16] to locally split up the preferences into secret shares. Increased security can be provided by introducing additional servers (with a minimum of two); the privacy of the data is guaranteed even if up to $n-1$ of n servers collude [16]. Next, the protocol distributes the sets of secret shares of all of these preferences to the servers such that no server has more than one secret share. At this point, each server has a database of effectively encrypted profiles stored as serialized arrays of secret shares. Once the servers collect enough of these user profiles (a minimum value k that can be configured at the time of deployment), the processing stage can commence.

3.2 Processing Stage

Recommendations are generated by aggregating the collection of all user profiles into a fixed set of generic representative profiles. These are recomputed and updated at regular intervals as new users specify their preferences and submit new profile data to the servers. The algorithm used for aggregation is an oblivious variant of the standard k-means clustering algorithm, adapted for use by multiple parties employing MPC. Each profile in the k-means algorithm is represented by a point in an n-dimensional space, where n is the number of preferences that make up one profile. The method for choosing initial cluster centers is an important distinction between this and the setup stages of other cluster analysis techniques. In a privacy-preserving scheme, choosing clusters by simple random selection introduces risks associated with choosing a cluster center with no nearby points.

In Figure 2, the leftmost mean was chosen outside the bounding box and has a weight of zero, which is problematic because we divide by the weight during the update step of k-means. We require that the points and weight values must remain secret, and MPC doesn't have capable divide-by-zero protection. We now briefly outline (later elaborated in §4.1) an alternative way to agree upon the initial means. All parties first compute a secret share of both the minimum and maximum point coordinate for each dimension. Next, they jointly compute the secret share of a

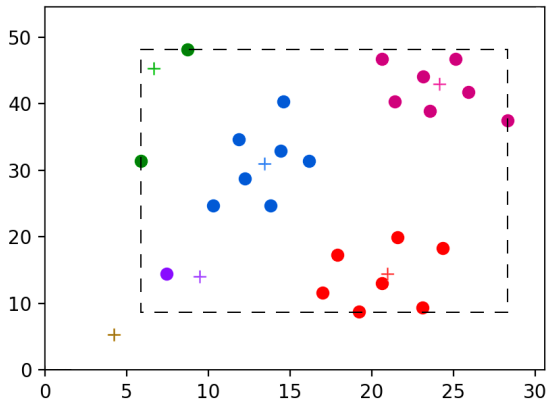

Figure 2: Clusters and initial means such that not every mean is assigned to a point.

random number, repeating this and using oblivious rejection sampling until a suitable initial mean coordinate has been generated for each dimension. This process ensures that all means are within the bounding box of the data, and we end up with k distinct clusters.

Execution of the two iterative steps of the clustering algorithm is relatively simple. Each point is assigned to its nearest mean by computing the pairwise Euclidean distances. Tie-breaking is not an issue from an implementation perspective, and choosing the minimum Euclidean distance does not require computing square roots because distances are only compared to other distances. Every mean is updated by computing a component-wise sum of all of the data points assigned to it and then dividing by the total weight of those points. These steps are repeated until there is reasonable confidence[1] that the means have converged.

3.3 Derivation and Delivery of Proposed Preference Profiles to Users

The newly calculated means (representing generic user profiles) are maintained by each server, and the one designated to give recommendations can use k-nearest neighbors to pick the mean closest to a particular partially completed preference profile submitted by a user. Note that the partial user profile is *not* revealed to the server during this process. The complete preference profile corresponding to that mean can then be sent as a recommendation to the user.

4 Implementation

In this section, we discuss in greater detail some of the challenges associated with implementing the overall service using MPC, and propose a suitable protocol for each component. A complete sample implementation for two servers, available online,[2] leverages the JIFF library [2].

4.1 Sampling within Secret Bounds.

When implementing efficient MPC protocols, it is helpful to minimize the number of special cases that must be tracked because MPC protocols cannot employ control flow or make branching decisions based on private data (without introducing exponential performance overheads). To avoid

[1] Because k-means clustering is executed at regular intervals and the means from each execution are maintained and reused for the next one, few iterations are necessary to generate satisfactory cluster means.

[2] A prototype implementation is available at `https://github.com/multiparty/user-privacy-preferences`.

the case of a mean without any assigned points, we require that all initial means are chosen within the bounding box of the data. One way to ensure this is with a rejection sampling algorithm such as the one in Figure 3.

```
label rejection_sample

share candidate = generate_random_share(length)  // share of random value
share range = secret_subtract(upper_bound, lower_bound)  // compute range
share lteq = secret_lteq(candidate, range)  // less than range

bool inbounds = await lteq.open()  // reveal validity

if inbounds
    return secret_add(lower_bound, candidate)
else
    goto rejection_sample
```

Figure 3: Rejection sampling algorithm.

Rejection Sampling Algorithm. This algorithm takes advantage of the ability to open/reveal whether a random candidate is in the bounding box, under the assumptions that (1) the initial means are only selected once before new data is added, and (2) it is intractable[3] to reconstruct the geometric random variable $\mathrm{G}(p)$ defined by the number of trials during sampling. The parameter $p = \frac{1}{\mathsf{upper}-\mathsf{lower}}$ would reveal the size of the bounding box if it were known.

Precompute-then-Choose. An alternative without the limitations outlined above is to precompute a random share of an integer for every potential upper bound.[4] The range can be computed as a secret share locally and then compared to each candidate range via equality computations under MPC. The result of each comparison is then multiplied by its corresponding candidate share and these products are summed and added to the lower bound to yield the final result. This approach costs a total of $|\mathbb{Z}_p|$ equality and multiplication operations (in parallel) and is only practical for applications in which the field $\mathbb{Z}_p$ used for secret shares is reasonably small. If the range is provided by only one of the parties (instead of existing only as a secret share), that party can share the range as a unit vector,[5] and the equality operations can be replaced with much cheaper dot product operations.

Additional Considerations. It is important to note that given only the ability to generate random integers in $[0, p)$ (easily done via Shamir's Secret Sharing), it is impossible to write a terminating algorithm that emits random numbers within another range $[0, n)$ where $n \leq p$ without knowing any additional information about n (such as $p \mod n$) [11]. This is unfortunate because in order to keep the input private, the running time of the algorithm must be the same across all possible inputs. An upper bound of infinity is obviously impractical, so we must depend on other techniques to generate a random integer in the range $[0, n)$ directly. Methods for generating random

[3] A malicious server would not be able to confidently guess the range of the data from the number of trials

[4] The distinction in this approach is that we can use regular rejection sampling because the upper bounds are public.

[5] As an example, if $p = 5$ and $\mathsf{range} = 4$ then the shared vector is $\langle 0, 0, 0, 1, 0 \rangle$.

integers in a nearly-uniform distribution exist that rely on secret modulo reduction. However, whether this option is reasonable varies according to the details of each use case and practical methods[6] to do so seem unjustifiably complicated for cases such as choosing initial means. Future research may lie in investigating discrete distribution generating (DDG) trees and hidden Markov models [10].

4.2 Split Databases

Servers must be online to receive new shares but only need to load past shares into memory when performing a joint secure computation to determine new means. Each secret share object created by JIFF has a party identifier and value (one point on the secret polynomial), a list of owners, and helper methods. All of these attributes (except for the value attribute) stay the same between each session in which means are recomputed, and are trivial to recreate. We take the share values of a received preference profile in the form of an array and store them in a JSON file. Before clustering, servers deserialize their partial databases into larger share objects.

4.3 k-Means Clustering

Our implementation of k-means achieves nearly the same results[7] as the standard non-secure version that operates on plaintext data. However, several intermediate steps require minor adjustments. Arithmetic operations on secret-shared values behave the same way in our protocol as they would on plaintext data, and for the averaging step we use a field that is at least large enough to accommodate p^2d distinct points with d preferences (*i.e.*, dimensions) and a maximum of p choices for any individual preference.

```
share distance, share[] distances
for mean in means
    for d in dimensions
        share half = mean[d].subtract(point[d])
        share full = half.multiply(half)
        distance = distance.add(full)
    distances.push(distance)

share min = distances[0]
share point.id = 0  // init as constant until if-else
for (i = 1; i < k; i++)
    share cmp = min.less_than(distances[i])
    min = cmp.if_else(min, distances[i])
    point.id = cmp.if_else(i, point.id)  // same as cmp*(x-y)+y
```

Figure 4: Point assignment algorithm.

Point Assignment. The assignment step shown in Figure 4 stores for each point p the value $\sum_d (p_d - m_d)^2$ for each dimension d of each mean m in an array and computes the minimum of this array to determine which mean is closest to the point. The operations involved are addition,

[6] Algesheimer *et al.* [3] requires computations that involve three different secret sharing schemes to improve cost beyond simple bit decomposition and long division.

[7] Division in our scheme has to be integer division, and final means can have minor rounding differences.

subtraction, and multiplication of secret-shared values. The oblivious ternary operator `if_else` is equivalent in cost to one multiplication.

```
share weight
for point in points
    for (i = 0; i < k; i++)
        share cmp = point.id.equals(i)
        weight = weight.add(cmp)
        for d in dimensions
            mean[d] = mean[d].add(cmp.multiply(point[d]))

for d in dimensions
    mean[d] = mean[d].div(weight)
```

Figure 5: Algorithm that updates cluster centers.

Recalculation of Means. The update step shown in Figure 5 computes the coordinate sum of all the points assigned to each mean and divides each sum by the total weight of those points to determine the new mean for the cluster. The operations involved are addition, subtraction, multiplication, and division of secret-shared values.

4.4 k-Nearest Neighbors

The k-nearest neighbors (k-NN) algorithm is conceptually straightforward to implement, especially in the discrete case where it is only necessary to count the number of equalities point-by-point and choose the maximum. However, this implementation has significant performance overheads because its cost is linear in the number of points and because equality operations are particularly inefficient under Shamir's Secret Sharing scheme (where even comparing a secret value to a constant value is expensive). By clustering the data in advance, we can reduce the amount of comparisons needed to use k-NN.

5 Evaluation

The four main factors that affect the running time of the k-means clustering stage are the number of data points (*i.e.*, profiles), the number of cluster means (*i.e.*, generic profiles), the number of iterations of the k-means algorithm, and the number of dimensions (*i.e.*, the maximum number of preferences in a profile).

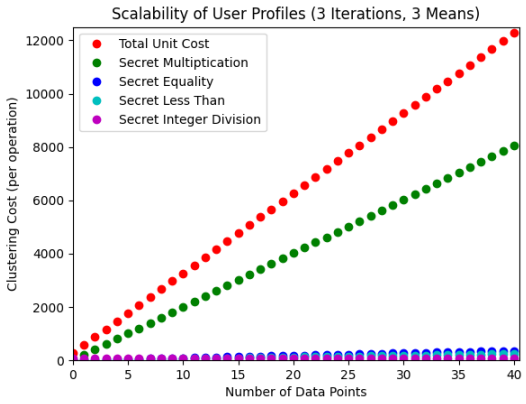

Figure 6: k-means clustering stage cost by number of profiles (3 iterations, 3 means).

As shown in Figure 6, the clustering cost is a little under 300 secret multiplication operations per point[8] while the number of means and iterations remain constant (here, at 3 each). A graph of the actual running time in seconds for each number of profiles would exhibit the same general pattern. In our prototype implementation, this corresponds to around 0.3 seconds per point when the protocol is executed between two servers that are communicating over the internet.

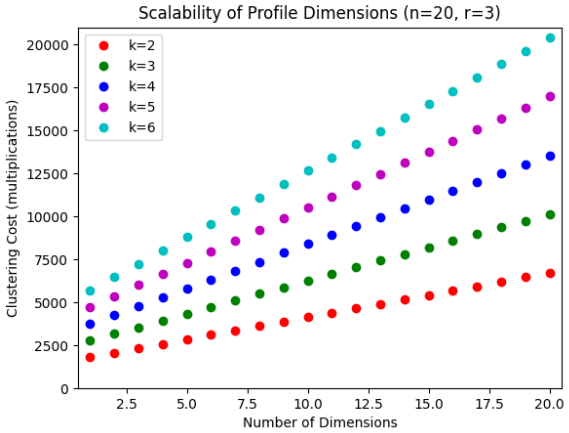

Figure 7: k-means clustering stage cost by number of profile dimensions (n=20, r=3).

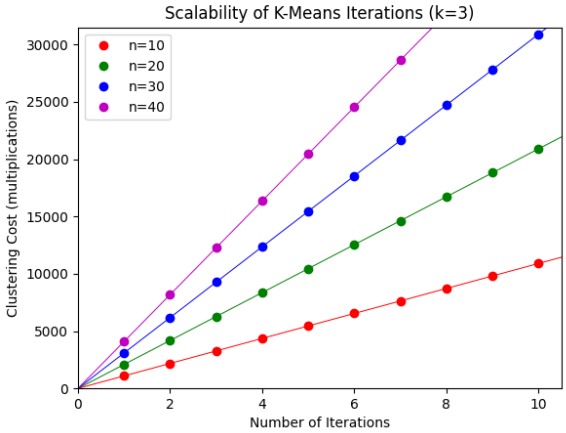

Figure 8: k-means clustering stage cost by number of k-means iterations ($k = 3$).

All other metrics (Figures 7 and 8) follow a linear trend. The relationship between the cost and the number of means appears to be linear, as well, based on the data in Figure 7.

5.1 Practical Performance Considerations

Take note that if the number of users who have submitted completed preference profiles is large, the cluster servers need not process all profiles after every submission, nor all old profiles after any submission. One option for reducing the amount of work performed by servers would be to treat the current means as wholly representatives of their respective clusters, and removing all old profiles assigned to those clusters. This is only necessary, however, after the data set becomes large enough to be a burden on the servers.

[8]See Section 5.2 for a discussion of an improvement on this bound.

Furthermore, all phases of the protocol are independent of each other. Once the first few users supply their profile data, anyone can receive a recommendation independently of when the data was last clustered (more recent data is preferable, but this is flexible). The servers can keep updating the cluster means while accepting new profiles, and users can submit their completed preferences at any time after receiving a recommendation.

The protocol also scales flexibly to n servers. However, do note that introducing more computation servers slows the computation overall[9] and is only useful for reducing the risk of collusion. As long as the servers remain separate from one another (and, ideally, are maintained by distinct organizations), the two-server configuration is the best option.

5.2 Speed-Up via Hybrid Protocols

For the purposes of this section, we define the *cost* as the number of multiplications involving two secret-shared multiplicands[10] and separate these multiplications (called the "Total Unit Cost" in Figure 6) into two categories: those called explicitly by the k-means algorithm and those called implicitly (*i.e.*, used internally) by the equality, comparison, or division operations on secret-shared values. The green plot in Figure 6 shows the number of multiplications *not* computed as part of higher-level operations, whereas the plots in Figures 7 and 8 only show the total number of multiplications of any kind. The sample implementation provided is the naive implementation of k-means on data that is secret-shared via Shamir's Secret Sharing scheme and a multiplication oracle[11].

Hybrid MPC frameworks such as JIFF, which we employ in our prototype implementation, provide the option to combine arithmetic circuits (computation over a prime field such as $\mathbb{Z}_p$) with Boolean circuits. JIFF implements the GMW [9] protocol, which lets an application developer implement algorithms using logical AND gates (or, in terms of arithmetic, multiplications) that have input and output values in $\mathbb{Z}_2$. JIFF's modular and extensible design allows it to leverage capabilities such as faster multiplication algorithms for $\mathbb{Z}_p$ via hybrid computation in $\mathbb{Z}_2$.[12]

Because k-means requires computing the Euclidean distance for each point, all explicit multiplications are necessary and independent—only the implicit multiplications in the comparisons or division can be reduced in number. This means that when there are exactly two clustering servers, the best running time we could hope to achieve is somewhere between the red and green dotted lines in Figure 6. Comparisons in Shamir's scheme have a running time that is quadratic in the number of parties, but yet scales linearly[13] on an array of bit shares. Thus, when there are more than two parties, GMW combined with arithmetic MPC can help flatten this growth in the number of comparison operations (as the number of comparisons *does not* remain linear in the scenarios with more than two servers). This optimization does not appear to have significant benefits in the two-party case, at least based on the data presented in the figures.

[9] Multiplication requires a number of rounds that is proportional to the number of parties, and comparisons take quadratically longer in purely polynomial schemes (see the discussion in Section 5.2).

[10] Multiplication by a constant, secret addition, and secret subtraction are implemented via a homomorphism in Shamir's Secret Sharing scheme, and thus are implemented using local computations that occur at each server. These do not require the servers to be online and to communicate, and so are not included in the cost calculations.

[11] Usually, this is implemented either using Beaver triples from a third party or via a BGW [4] protocol

[12] JIFF implements this optimization via composition of arithmetic shares into bit-wise GMW shares, followed by binary multiplication, and finally conversion from $\mathbb{Z}_2$ back into arithmetic shares within $\mathbb{Z}_p$.

[13] This is incorporated into a development version of JIFF, with a cost of 1 AND gate per bit per party to compose GMW shares into arithmetic shares.

6 Conclusions and Future Work

Our evaluation and associated benchmarks support the conclusion that the recommendation protocol can scale to a large number of users and preference choices while remaining practical to operate. Although our use case involved helping users navigate the process of specifying their privacy preferences, this type of protocol can easily be generalized to fit a much wider array of scenarios where it may be beneficial to generate data-driven recommendations automatically but in a privacy-preserving way. The specific problem of generating uniformly random integers within secret bounds as discussed in Section 4.1 is of interest in its own right and efficient solutions would be useful in a variety of other algorithms that might be implemented to run under MPC. Broader topic areas for future work include exploration of alternative privacy-preserving machine learning approaches, implementation of other privacy-preserving clustering algorithms, introduction of better support for common linear algebra algorithms within MPC frameworks such as JIFF, and construction of frameworks that make it possible to perform tasks such as solving satisfiability problems under MPC.

Acknowledgments

The work described in this report has been partially supported by the NSF (under Grants #1430145, #1414119, #1718135, and #1739000) and the Honda Research Institutes.

References

[1] The California Consumer Privacy Act of 2018. `https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180AB375`.

[2] K. D. Albab, R. Issa, A. Lapets, P. Flockhart, L. Qin, and I. Globus-Harris. Tutorial: Deploying Secure Multi-Party Computation on the Web Using JIFF. In *2019 IEEE Cybersecurity Development (SecDev)*, pages 3–3, McLean, VA, USA, September 2019.

[3] J. Algesheimer, J. Camenisch, and V. Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *Annual International Cryptology Conference*, pages 417–432. Springer, 2002.

[4] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 351–371. 2019.

[5] A. Bestavros, A. Lapets, and M. Varia. User-Centric Distributed Solutions for Privacy-Preserving Analytics. *Communications of the ACM*, 60(2):37–39, February 2017.

[6] H. Corrigan-Gibbs and D. Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. *CoRR*, abs/1703.06255, 2017.

[7] Council of European Union. Council regulation (EU) no 2016/679, 2016. `https://eur-lex.europa.eu/legal-content/EN/TXT/?qid=1589662968663&uri=CELEX:32016R0679`.

[8] S. Das, L. A. Dabbish, and J. I. Hong. A typology of perceived triggers for end-user security and privacy behaviors. In *Proceedings of the Fifteenth USENIX Conference on Usable Privacy and Security*, SOUPS'19, pages 97–115, USA, 2019. USENIX Association.

[9] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 307–328. 2019.

[10] J.-P. Katoen. The probabilistic model checking landscape. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 31–45. ACM, 2016.

[11] D. E. Knuth and A. C. Yao. The complexity of nonuniform random number generation. *Algorithm and Complexity, New Directions and Results*, pages 357–428, 1976.

[12] A. Lapets, F. Jansen, K. D. Albab, R. Issa, L. Qin, M. Varia, and A. Bestavros. Accessible Privacy-Preserving Web-Based Data Analysis for Assessing and Addressing Economic Inequalities. In *Proceedings of ACM COMPASS 2018: First Conference on Computing and Sustainable Societies*, San Jose, CA, USA, June 2018.

[13] J. Lemley, S. Bazrafkan, and P. Corcoran. Deep learning for consumer devices and services: Pushing the limits for machine learning, artificial intelligence, and computer vision. *IEEE Consumer Electronics Magazine*, 6(2):48–56, 2017.

[14] L. Qin, P. Flockhart, A. Lapets, K. D. Albab, M. Varia, S. Roberts, and I. Globus-Harris. From Usability to Secure Computing and Back Again. In *Proceedings of the 15th Symposium on Usable Privacy and Security (SOUPS)*, Santa Clara, CA, USA, August 2019.

[15] S. Ruoti, J. Andersen, D. Zappala, and K. E. Seamons. Why johnny still, still can't encrypt: Evaluating the usability of a modern PGP client. *CoRR*, abs/1510.08555, 2015.

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

[17] F. Wang, R. Ko, and J. Mickens. Riverbed: Enforcing user-defined privacy constraints in distributed web services. In *NSDI*, pages 615–630, 2019.

[18] A. Whitten and J. D. Tygar. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *Proceedings of the 8th Conference on USENIX Security Symposium - Volume 8*, SSYM'99, page 14, USA, 1999. USENIX Association.

[19] A. C. Yao. Protocols for Secure Computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society.

[20] E. Zeng and F. Roesner. Understanding and improving security and privacy in multi-user smart homes: A design exploration and in-home user study. In *Proceedings of the 28th USENIX Conference on Security Symposium*, SEC'19, pages 159–176, USA, 2019. USENIX Association.