Efficient Scalable Multiparty Private Set-Intersection via Garbled Bloom Filters

Roi Inbar[1]*, Eran Omri[1]†, and Benny Pinkas[2]‡

[1] Department of Computer Science, Ariel University, Israel
roikeman@gmail.com, omrier@ariel.ac.il
[2] Department of Computer Science, Bar-Ilan University, Israel
benny@pinkas.net

Abstract. In private set intersection (PSI), a set of parties, each holding a private data set, wish to compute the intersection over all data sets in a manner that guarantees both correctness and privacy. This secure computation task is of great importance and usability in many different real-life scenarios. Much research was dedicated to the construction of PSI-tailored concretely efficient protocols for the case of two-party PSI. The case of many parties has been given much less attention, despite probably being a more realistic setting for most applications.

In this work, we propose a new concretely efficient, highly scalable, secure computation protocol for multiparty PSI. Our protocol is an extension of the two-party PSI protocol of Dong et al. [ACM CCS'13] and uses the garbled Bloom filter primitive introduced therein. There are two main variants to our protocol. The first construction provides semi-honest security. The second construction provides (the slightly weaker) augmented semi-honest security, and is substantially more efficient. Furthermore, in the augmented semi-honest protocol all heavy computations can be performed ahead of time, in an offline phase, before the parties ever learn their inputs. This results in an online phase that requires only short interaction. Moreover the in the online phase, interactions are performed over a star topology network. All our constructions tolerate any number of corruptions.

We implemented our protocols and incorporated several optimization techniques. These techniques allow the running time of the protocol to be comparable to that of the two party protocol of Dong et al. and scale linearly with the number of parties. We ran extensive experiments to compare our protocol with the two-party protocol and to demonstrate the effect of the different optimizations.

Keywords: Multiparty computation; Private set intersection; Concrete efficiency; Garbled Bloom filters.

*Research supported by ISF grant 544/13.

†Research supported by ISF grants 544/13 and 152/17.

‡Research supported by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minsters Office, and by ISF grant 1018/16.

1 Introduction

Powerful feasibility results for secure multiparty computation were given three decades ago [23, 10, 1] demonstrating that any polynomial time computable function can also be *securely* computed. Furthermore, in the last decade, there has been tremendous progress in the construction of concretely efficient generic secure two-party protocols, and for extent also for the multiparty setting.

Most of the above progress in concretely efficient secure computation was made in the design of generic protocols via the circuit evaluation paradigm, which allows parties to jointly and efficiently compute a logical or arithmetic circuit for computing the functionality at hand. The generic approach, however, becomes much less applicable for functionalities that require the evaluation of large circuits. One such example is *private set intersection* (PSI), which is the focus of this work.

In the PSI problem a set of parties, each holding a large private data set, wish to compute the intersection over all data sets. PSI is of great relevance to many different real-life scenarios, motivated, for example by the need to perform joint computational tasks over several sensitive databases. Much research was dedicated to the construction of PSI-tailored highly efficient protocols for the case of two parties. A survey of the abundance of works on efficient two-party PSI protocols is given in [20], including a classification of the underlying techniques. Some results on this topic can be found in, e.g., [7, 13, 5, 6, 16, 19, 8, 20, 17], where many of the recent results include optimized implementations. To our discussion, most relevant is the work of [6], which we describe in detail below.

The case of more than two parties was given much less attention. To the best of our knowledge the only implementation of a concretely efficient multiparty PSI protocol to date was recently given in [18]. Nevertheless, multiparty private set intersection remains a very relevant and important question and the case of many parties is the right setting in many scenarios. As a running example, we take the scenario that motivated this work to begin with. Consider a set of governmental or commercial agencies wishing to collaborate to detect a possible intrusion attack to a common network. Each agency must protect the privacy of its information and of its costumers. However, as part of the collaborative effort to detect an intrusion, the agencies are interested in finding the intersection over the sets of suspicious IP addresses held by each agency.

In light of the above, the main question that this work deals with is:

> Construct concretely efficient secure multiparty protocols for computing private set intersection that scale well with the number of parties and with data set size.

1.1 The Protocol of Dong et al. [6]

The starting point of our work is the two-party PSI protocol of [6]. They introduced *garbled Bloom filters* – a cryptographic variant of the Bloom filter data structure, introduced by Bloom [3]. Recall that a Bloom filter, $\mathrm{BF}_{\mathcal{S}}$, encodes a set $\mathcal{S}$ of elements as an m-bit vector with respect to k randomly selected hash functions $h_1, \ldots, h_k$. To insert an element x into the Bloom filter, the indices $h_1(x), \ldots, h_k(x)$ are all set to 1 (all indices are initialized to 0). A search query is never answered by a false negative, and is answered by a false positive with overwhelmingly low probability for the right choice of k and m (depending on the bound on the size of the set).

Dong et al. [6] introduced a garbled version of the Bloom filter (GBF), obtained by expanding each bit in the original Bloom filter to a λ-long bit string (where λ depends on the security parameter). The strings are chosen such that for every x, if $x \in \mathcal{S}$, then the XOR of the strings in indices $h_1(x), \ldots, h_k(x)$ is the all-zero string[3], and is a uniformly chosen string otherwise (i.e., if $x \notin \mathcal{S}$). The false negative probability of GBF searches is inherited from the false positive probability of the original Bloom filter. The false positive probability is $2^{-\lambda}$.

A property of garbled Bloom filters which is very useful for computing the intersection, is that for two sets $\mathcal{S}_1, \mathcal{S}_2$ the bit-wise XOR on $\mathrm{GBF}_{\mathcal{S}_1}$ and $\mathrm{GBF}_{\mathcal{S}_2}$ yields $\mathrm{GBF}_{\mathcal{S}_1 \cap \mathcal{S}_2}$. In addition, seeing the strings in the GBF for any proper subset the indices $h_1(x), \ldots, h_k(x)$ leaks *nothing* on whether $x \in \mathcal{S}$ or not.

The construction of [6] considered a client C and a server S and worked in the semi-honest model (which we also consider here). In a preliminary phase, the parties agree on a sequence of hash functions (modeled as random functions). The client, holding a set $\mathcal{S}_{\mathsf{C}}$, is instructed to construct a local $\mathrm{GBF}_{\mathcal{S}_{\mathsf{C}}}$ and the server S, holding a set $\mathcal{S}_{\mathsf{S}}$, is instructed to construct a local $\mathrm{BF}_{\mathcal{S}_{\mathsf{S}}}$ (both, with respect to the predetermined set of hash functions). Then, using oblivious transfer, for each $i \in [k]$ (where k is the size of the filters), S learns a string s_i from C, where if $\mathrm{BF}_{\mathcal{S}_{\mathsf{S}}}[i] = 1$ it holds that $s_i = \mathrm{GBF}_{\mathcal{S}_{\mathsf{C}}}[i]$ and s_i is a randomly selected string otherwise. The security of the oblivious transfer ensures that the clients learn nothing about the choices of the server, and the server learns nothing about the value of $\mathrm{GBF}_{\mathcal{S}_{\mathsf{C}}}[i]$ whenever $\mathrm{BF}_{\mathcal{S}_{\mathsf{S}}}[i] = 0$. By the properties of garbled Bloom filters, the server ends up with the garbled Bloom filter of the intersection.

Recently, Rindal and Rosulek [21] extended the construction of [6] to the malicious setting, using cut-and-choose techniques. We believe that similar techniques may be applied to our constructions to obtain a maliciously secure multiparty PSI protocol. We leave this as future work.

1.2 Efficient secure PSI for many parties

Freedman et al. [7] suggested a multiparty PSI protocol, based on oblivious polynomial evaluation (OPE) which is implemented using additively homomorphic encryption, such as Paillier encryption scheme. Recently, Hazay and Venkitasubramaniam [11] presented a reduction from the multiparty (semi-honest and malicious) case to the two-party case. Specifically, they run a version of the protocol of [7] between pairs of parties. Their construction runs over a star network

[3] In the original work of [6], this value was x itself, rather than the all-zero string. This change is of no real importance, however, it makes the presentation of our construction simpler.

topology and is asymptotically efficient. However, it requires a linear number of encryptions and decryptions of an additively homomorphic public-key encryption scheme.

The only work, we are aware of, that offered an implementation of a concretely efficient secure multiparty PSI protocol is the very recent work of Kolesnikov et al. [18]. They propose a highly efficient construction, based on a new primitive that they call oblivious programmable pseudorandom function.

1.3 Our Contribution

In this work we extend the construction of [6] to obtain protocols for securely computing the PSI functionality with many parties. We describe three protocols for three different settings, differing on the assumed adversarial model. All our protocols are highly efficient and scale well as the number of parties and the size of each data set grow. We implemented all three constructions and ran extensive experiments to evaluate the different components of our protocols and the multiple improvements that were incorporated. The experiments also nicely demonstrate the scalability of our construction. We next describe the three variant of the protocol and motivate each of them.

An information theoretic construction – when the server is external and does not collude with other parties. Consider our running example of collaborative intrusion detection, and consider the case where a regulator (server), which is assumed to never collude with any of the clients, wishes to learn the intersection over all the data sets of local agencies (clients). For this case, we construct a protocol that uses no cryptographic hardness assumptions for securely computing the intersection over the data sets of $t-1$ clients $\mathsf{P}_1, \ldots, \mathsf{P}_{t-1}$. Indeed, this protocol is the basis of all our constructions.

The protocol is initialized by the server randomly choosing a sequence of k hash functions $h_1, \ldots, h_k$ and sending their description to the clients. Each client P_i first locally computes a garbled Bloom filter GBF_i encoding its private data set (with respect to $h_1, \ldots, h_k$). The client then selects t random strings $s_1^i, \ldots, s_{t-1}^i$, each as long as the GBF, under the constraint that the XOR of these t strings equals GBF_i (i.e., it is a t-out-of-t XOR secret sharing of GBF_i). Finally, each client XORs all the shares it received (i.e., client j computes the XOR of $s_j^1, \ldots, s_j^{t-1}$) to obtain a share s_j^* of the garbled Bloom filter of the intersection. The client then sends the result to the server P_0. The server computes the XOR of all the shares it received to obtain the resulting garbled Bloom filter.

The correctness of the protocol follows from the fact that the XOR of two garbled Bloom filters is a garbled Bloom filter of the intersection. The security of the protocol stems from the fact that all that clients see are random shares, and the server learns nothing but the XOR of all local GBFs. See the full version of our paper for the formal description of the protocol.

A semi-honest construction. Before demonstrating how to extend the previous construction to provide semi-honest security, let us point out the changes in the settings and the shortcomings of the information theoretic construction. The first change is that the server should not learn anything about the intersection of the data sets of $\mathsf{P}_1, \ldots, \mathsf{P}_{t-1}$ (other than what is implied by the intersection of all parties). This could be overcome by a similar manner to what is done in the two-party protocol of [6]. That is, the server locally computes a Bloom filter BF_0 and for every coordinate ℓ, if $\mathrm{BF}_0[\ell] = 1$ then P_0 asks from each client P_i the ℓ'th block of the share s_j^*, and if $\mathrm{BF}_0[\ell] = 0$, then P_0 asks from each client P_i a random string instead. Let $s^* = \mathrm{GBF}_{\mathcal{IS}}$ be the secret filter reconstructed by the server P_0. Using the oblivious transfer functionality, the interaction is done such that P_i learns nothing about the choice of the server, and the server learns nothing about the string it has not chosen to learn. We denote the communication pattern of this interaction as the *star* protocol.

The second change is in the adversarial model. Before, we assumed that if the adversary corrupts the server P_0, then it does not corrupt any of the clients. Now, we put no such restriction on the adversary. Consider the case that an element x is an element in the set of P_0 but not in the set of P_1. In this case, the adversary corrupting P_0 and P_1, must not learn whether x is in the intersection of the sets of all honest parties $\mathsf{P}_2, \ldots, \mathsf{P}_{t-1}$ or not. In the star protocol, however, an adversary controlling P_1 can XOR GBF_1 with the final output of P_0 to obtain the intersection of all honest parties together with P_0. The server therefore learns whether x is in this intersection, which occurs if and only if x is in the intersection of the sets of all honest parties.

To solve this problem, we instruct each pair of parties to exchange shares by engaging in an oblivious transfer interaction, where each party P_i asks P_j for a random string as the ℓ'th block, whenever $\mathrm{BF}_i[\ell] = 0$ and $s_j^*[\ell]$ otherwise. This ensures that if x is not in the intersection (and specifically $\mathrm{BF}_{i'}[\ell] = 0$ for some i'), then all parties contribute 'noise' to $\mathrm{GBF}_{\mathcal{IS}}[\ell]$.

An augmented-semi-honest construction. An augmented semi-honest adversary is the same as a semi-honest one, with the only difference that it can choose any probable input. It should be noticed that in our settings, this strengthens the simulator more than it does the real-world adversary. Thus, this security definition is actually easier to obtain (than semi-honest). In this paper, we show that our star protocol already guarantees augmented semi-honest security. The intuition for that is that we can simulate the protocol by selecting (in the ideal model) the input of all corrupted parties to be the same as the input of the P_0.

1.4 Optimizations to the augmented semi-honest protocol

Computation. All the heavy computations of the augmented semi-honest protocol can be performed ahead of time, before the parties learn their inputs. This is done using two main ideas. The first is the OT-extension paradigm of [12], allowing the computation intensive part of the oblivious transfer interactions to be performed ahead of time. Second, we observe that the secret sharing of the local GBFs can also be done ahead of time, by having each party simply send random shares to all other parties in the offline phase, and then adjusting its local share to be the XOR of all the other shares and the GBF (constructed upon receiving the input data set). In this manner, the online phase only consists of a short interaction between the server and each of the other parties.

Communication. The goal of this optimization is to load balance the interactions between pairs of parties. Instead of having all parties send messages directly to the server, it is possible for them to route the messages through their peers, over a hypercube network structure. Each party which receives messages, aggregates them (namely, computes the XOR of the received GBFs) before forwarding the result in the direction of the server. It turns out that using this method, we were able to decrease the linear number of OT interactions of the server, to a logarithmic number of interactions for any party, and this improve the overall latency of the protocol. We elaborate on this optimization in Section 4.1.[4]

1.5 Implementations and Experimental Results

We implemented all of our protocols. Our code is based on the open source code of [6]. Nevertheless, we incorporated several improvements and techniques that allow the protocol in the multiparty setting to be linearly dependent on the number of parties and the data sets size (as one would expect from a theoretical analysis). In Section 4, we describe our implementation and optimizations, and detail the experiments that we ran.

Our implementations may be compared with those of [18]. We use the measurements reports from their paper to compare. Evidently, for small numbers of parties their implementation outperforms ours. It is our understanding that the reason for that is twofold. First, we think that our code can be improved and in particular rewritten in C++, rather than Java (which is currently the case). We believe that this change alone would result in an improvement by a factor of 2. Second, the GBF based construction comes with an inherent cost in communication complexity. When the number of parties grows, however, our protocols seem to gain on that of [18]. The protocol of [18] contains a phase in which a quadratic number of comparisons are made. Indeed, while we report on experiments with up to 56 parties for our augmented semi-honest construction, [18] only report on experiments with at most 15 parties. Our experimental results show a very slow growth in running time as the number of parties grow (sub-linear), see Table 4.

We summarize the theoretical overhead of our three constructions in Table 1. Therein m is the BF size, λ is the GBF bit-string length (i.e., the GBF Size is λm), t is the number of parties. The table is split into P_0 (the server that learns the output) and P_i, playing the role of a client with no output. "Hashes Used" counts the number of HashRange accesses; "Memory Complexity" refers to the worst case memory require to store GBFs; "Communication Complexity"

[4] Originally, the hypercube method [2] was used to speed up message propagation replacing a star like propagation scheme with a tree like scheme. We use it in order to aggregate messages sent by all parties to the server.

count the number of bits that will sent or received; “OT Extensions” counts the number of OT Protocol invocations; “Create Shares” counts PRG accesses.

	MPSI		***MPSI*-Aug**		***MPSI*-NoOT**	
Operation	P_0	P_i	P_0	P_i	P_0	P_i
Hashes Used	$2(k \cdot n)$	$2(k \cdot n)$	$2(k \cdot n)$	$(k \cdot n)$	$(k \cdot n)$	$(k \cdot n)$
Memory Complexity	$t(\lambda+1)m$	$t(\lambda+1)m$	$t(\lambda+1)m$	$(\lambda+1)m$	$t(\lambda+1)m$	$(\lambda+1)m$
Communication Complexity	$\lambda \cdot m \cdot t$	$\lambda \cdot m \cdot t$	$\lambda \cdot m \cdot t$	$\lambda \cdot m$	$\lambda \cdot m \cdot t$	$\lambda \cdot m$
OT Extensions	$m \cdot t$	$m \cdot t$	$m \cdot t$	m	–	–
Create Shares	$2\lambda mt$	$2\lambda mt$	–	$2\lambda m(t-1)$	–	$2\lambda m(t-1)$
	With Hyper Cube Communication					
Hashes Used	–	–	$2(k \cdot n)$	$2(k \cdot n)$		
Memory Complexity	–	–	$\log(t)(\lambda+1)m$	$\log(t)(\lambda+1)m$	$\log(t)(\lambda+1)m$	$\log(t)(\lambda+1)m$
Communication Complexity	–	–	$\lambda \cdot m \cdot \log(t)$	$\lambda \cdot m \cdot \log(t)$	$\lambda \cdot m \cdot \log(t)$	$\lambda \cdot m \cdot \log(t)$
OT Extensions	–	–	$m \cdot \log(t)$	$m \cdot \log(t)$	–	–
Create Shares	–	–	–	$2\lambda m(t-1)$	–	$2\lambda m(t-1)$

Table 1: Theoretical complexity analysis.

Protocol	**Communication**		**Computation**		**Security Model**
	Leader	Client	Leader	Client	
KMPRT17 [18]	$\mathcal{O}(tn\lambda)$	$\mathcal{O}(tn\lambda)$	$\mathcal{O}(t\kappa)$	$\mathcal{O}(t\kappa)$	semi-honest
Here	$\mathcal{O}(tn\lambda k)$	$\mathcal{O}(tn\lambda k)$	$\mathcal{O}(\lambda ntk)$	$\mathcal{O}(\lambda ntk)$	semi-honest
KMPRT17 [18]	$\mathcal{O}(tn\lambda)$	$\mathcal{O}(n\lambda)$	$\mathcal{O}(t\kappa)$	$\mathcal{O}(\kappa)$	augmented semi-honest
Here	$\mathcal{O}(tn\lambda k)$	$\mathcal{O}(n\lambda k)$	$\mathcal{O}(\lambda ntk)$	$\mathcal{O}(\lambda ntk)$	augmented semi-honest
Here (hypercube)	$\mathcal{O}(\log(t)n\lambda k)$	$\mathcal{O}(\log(t)n\lambda k)$	$\mathcal{O}(\lambda ntk)$	$\mathcal{O}(\lambda ntk)$	augmented semi-honest

Table 2: Theoretical complexity analysis – in comparison to state of the art.

1.6 More Related Work

Much research was dedicated to the construction of PSI-tailored highly efficient protocols for the case of two-party set intersection. A survey on efficient two-party PSI protocols is given in Pinkas et al. [20], including a classification of the underlying techniques. Public-key based PSI protocols were presented in, e.g., [7, 8, 5] and the oblivious-transfer based and oblivious-pseudo-random-function based PSI protocols (see, e.g., [6, 16, 19]).

2 Preliminaries

For space considerations we only describe here some less standard definitions. For $n \in \mathbb{N}$, let $[n] = \{1, \ldots, n\}$. Given a random variable (or a distribution) X, we write $x \leftarrow X$ to indicate that x is selected according to X. We use the abbreviation PPT to denote probabilistic polynomial-time. All polynomials that we will consider will be with respect to the security parameter, unless explicitly stated otherwise; specifically, all polynomial time machines will be polynomial in the security parameter.

2.1 Secure Multiparty Computation and the MPSI Functionality

We follow the standard definitions of secure multiparty computation for semi-honest adversaries according to the ideal versus real paradigm (cf. [9]). All parties run in probabilistically polynomial time, and adversaries are non-uniform. We consider *semi-honest adversaries* who follow the prescribed protocol faithfully, but may try to infer additional information about the honest parties as the protocol terminates. We also consider *augmented semi-honest adversaries*, which are similar to semi-honest ones, with the only difference bein that such adversaries are allowed to change their input to any other (valid) input. We next give the definition of the MPSI functionality.

Definition 1 (multiparty private set intersection).
Functionality $\mathcal{F}_{MPSI}$:
Inputs: *All parties hold the number of parties t, an upper bound M on the number of elements in any data set, and the security parameter κ. In addition, each party* P_i *has a data set* $\mathcal{DB}_i$ *as its private input.*
Computation: *Compute the intersection of all data sets, i.e.,* $\mathcal{IS} = \bigcap_{i=0}^{t} \mathcal{DB}_i$*.*
Outputs: *Party* P_0 *receives* $\mathcal{IS}$ *from the functionality, and all other parties receive no output.*

Bloom Filters. Bloom filters were introduced by Bloom [3] as a compact data structure for probabilistic set membership testing. A Bloom filter encodes a subset $\mathcal{S}$ of elements in some domain $\mathcal{D}$ into an array of m bits, where each element in the domain is attributed with a subset of the indices in the bit array. Specifically, a Bloom filter is parametrized by a sequence of k hash functions $H = (h_1, \ldots, h_k)$, and an element x is attributed with the indices $(h_1(x), \ldots, h_k(x))$. To encode a set of elements $\mathcal{S}$, all the bits in the array with index that is attributed to some $x \in \mathcal{S}$ are set to 1 and all other bits are set to 0.

It is easy to verify that for two sets $\mathcal{S}_1, \mathcal{S}_2$ that were encoded (with the same H) into $\mathrm{BF}_1, \mathrm{BF}_2$ it holds that $\mathrm{BF}_1 \otimes \mathrm{BF}_2$ encodes $\mathcal{S}_1 \cap \mathcal{S}_2$, where $\otimes$ is the bit-wise AND operator. This feature will play an important role in the constructions introduced in this work, and a variant thereof (where the bitwise AND is replaced with a bitwise XOR) will apply to the garbled variant of Bloom filters that will be used here.

Garbled Bloom Filters. A garbled variant of Bloom filters (GBF) was introduced by Dong et al. [6]. The garbled version of a Bloom filter is obtained by expanding each bit in the original Bloom filter to a long bit string (whose length depends on the security parameter). The compactness of the original Bloom filter is somewhat compromised here for the sake of obtaining an obliviousness property. Intuitively, this obliviousness property means that for a given element x, it is impossible to learn anything on whether x is in the data set without querying the GBF on *all* indices attributed to x.

GBF is an array of $m \in \mathbb{N}$ bit strings, each of length $\lambda \in \mathbb{N}$. Similarly to a Bloom filter, a GBF is parametrized by a sequence of k hash functions $H = (h_1, \ldots, h_k)$. To insert an item x, where $j_1, \ldots, j_k$ are the k indices attributed to x, first choose a vacant index $\mathsf{finalInd}$ (namely, $\mathsf{finalInd}$ is not attributed to any element x' previously inserted to the GBF). Second, treat all other indices j_i. If j_i is also vacant, then set it to a randomly chosen λ long bit string. Otherwise, do noting (the appropriate string was previously determined). Finally, set the string at index $\mathsf{finalInd}$ to the bit-wise XOR of all other $k-1$ strings.

3 Multiparty PSI Protocols

3.1 A Protocol with Semi-Honest Security

We describe our construction of a semi-honest secure multiparty set intersection protocol. As in all our constructions, the key idea is to let the parties jointly compute the garbled Bloom filter for the intersection of all data sets. Recall that for two sets $\mathcal{S}_1, \mathcal{S}_2$ that were encoded (with the same H) into $\mathrm{GBF}_1, \mathrm{GBF}_2$ it holds that $\mathrm{GBF}_1 \oplus \mathrm{GBF}_2$ encodes $\mathcal{S}_1 \cap \mathcal{S}_2$, where $\oplus$ is the bit-wise XOR operator. The formal description of the protocol appears in Figure 1.

We prove the security of the protocol, formally given in the following theorem, in the full version of the paper.

Theorem 1. *Protocol* MPSI *(appearing in Figure 1) computes* $\mathcal{F}_{\mathrm{MPSI}}$ *with statistical security in the* $\mathcal{F}_{OT}$*-hybrid model,*[5] *in the semi-honest model, for the right choice of parameters* m *and* k *as functions of the security parameter* κ *and the bound* M *on the size of each individual data set.*

The composition theorem of [4], immediately yields the following corollary.

Corollary 1. *Assume trap-door permutations exist. Then, protocol* MPSI *(appearing in Figure 1) securely computes* $\mathcal{F}_{\mathrm{MPSI}}$ *in the semi-honest model, for the right choice of parameters* m *and* k *as functions of the security parameter* κ *and the bound* M *on the size of each individual data set.*

[5] We stress that as we run many instantiations of $\mathcal{F}_{\mathrm{OT}}$ in parallel, we need to use an OT protocol that is secure under parallel composition.

Semi-Honest Protocol

Input: All parties in the set $\mathcal{P} = \{\mathsf{P}_0, \ldots, \mathsf{P}_{t-1}\}$ hold the security parameter κ, the size t of $\mathcal{P}$, and the parameters m, k, and λ – for the Bloom filter length, the number of hash functions, and the length of each string in the GBF, respectively. In addition, each party P_i holds its private data set $\mathcal{DB}_i$.

Initialization: P_0 randomly selects a sequence $H \subseteq \mathcal{H}$ of k hash functions, and sends its description (i.e., keys) to all other parties.

Local GBF**s generation:** Each $\mathsf{P}_i \in \mathcal{P}$ acts as follows:

1. Creates a local $\mathrm{BF}_{\mathcal{DB}_i} = \mathsf{BuildBF}(\mathcal{DB}_i, k, m, H)$.
2. Creates a local $\mathrm{GBF}_{\mathcal{DB}_i} = \mathsf{BuildGBF}(\mathcal{DB}_i, k, m, H, \lambda)$.
3. Selects uniform strings (shares) $\left\{\mathrm{GBF}^{j}_{\mathcal{DB}_i}\right\}_{j=0}^{t-1}$ conditioned on $\bigoplus_{j=0}^{t-1} \mathrm{GBF}^{j}_{\mathcal{DB}_i} = \mathrm{GBF}_{\mathcal{DB}_i}$.[a]

***OT* interactions:** Each $\mathsf{P}_i \in \mathcal{P}$ (taking the role of the receiver) engages with each $\mathsf{P}_j \in \mathcal{P} \backslash \mathsf{P}_i$ (taking the role of the sender) in an OT interaction for each $\ell \in [m]$, as follows.

- Party P_j uses $s_0^{(i,j,\ell)} \leftarrow \{0,1\}^\lambda$ and $s_1^{(i,j,\ell)} = \mathrm{GBF}^{j}_{\mathcal{DB}_i}[\ell]$ as its input strings to the OT.
- Party P_i uses $\mathrm{BF}_i[\ell]$ as input bit to the OT.

Denote by ${\mathrm{GBF}^{\star}}^{i}_{\mathcal{DB}_j}$ the vector whose entry ${\mathrm{GBF}^{\star}}^{i}_{\mathcal{DB}_j}[\ell]$ is the output of party P_i in the ℓ'th OT interaction with P_j.

Obtaining final $\mathcal{IS}$ shares: Each $\mathsf{P}_i \in \mathcal{P}$ constructs its share of the intersection GBF by computing ${\mathrm{GBF}^{\star}}^{i}_{\mathcal{DB}_{\mathcal{IS}}} = \bigoplus_{j=0}^{t-1} {\mathrm{GBF}^{\star}}^{i}_{\mathcal{DB}_j}$.

Output reconstruction by P_0: Each party P_i sends it share ${\mathrm{GBF}^{\star}}^{i}_{\mathcal{DB}_{\mathcal{IS}}}$ to P_0, which in turn, computes $\mathrm{GBF}_{\mathcal{IS}} = \bigoplus_{i=0}^{t-1} {\mathrm{GBF}^{\star}}^{i}_{\mathcal{DB}_{\mathcal{IS}}}$. Party P_0 computes $\mathcal{IS} = \{x \in \mathcal{DB}_0 \colon \mathrm{GBFQuery}(\mathrm{GBF}_{\mathcal{IS}}, x, m, k, H, \lambda) = 1\}$.

Output: P_0 outputs $\mathcal{IS}$.

[a] This could be replaced by any other t-out-of-t secret sharing scheme.

Fig. 1: Protocol MPSI – multiparty private set intersection with semi-honest security

3.2 A Protocol with Augmented Semi-Honest Security

We describe the protocol GBF-MPSI-aug that is secure against *augmented* semi-honest adversaries. Recall that an augmented semi-honest adversary must follow the protocol honestly, but is also allowed to select a different input (from the correct domain) upon engaging in a protocol execution. On the face of it, this may seem as a stronger definition of security, since the real model adversary is more powerful than a semi-honest one. However, it turns out that it is actually easier to obtain in the case of multiparty set intersection (with a single output). The intuition for this is that the definition also empowers the ideal model adversary by allowing it to select different inputs, which it is unable to do in the ideal semi-honest setting.

Indeed, the protocol we describe in this section is a (faster) variant of the protocol that was introduced in Section 3.1. The main change here is that all

OT interactions are performed in a star-like communication graph (rather than a complete network communication graph), with the server P_0 taking the role of the receiver.

The key idea of the construction is to first let parties $\mathsf{P}_1, \ldots, \mathsf{P}_{t-1}$ jointly compute the garbled Bloom filter for the intersection of their data sets (without the data set of P_0). Then, each party P_i interacts with the server P_0 via an oblivious transfer for each entry in P_i's (share of the) GBF, such that P_0 receives the real share part only for those entries that are attributed to elements that P_0 holds (and random strings otherwise). The formal description of the protocol appears in Figure 2.

Theorem 2. *Protocol* MPSI-*Aug (appearing in Figure 2) computes* $\mathcal{F}_{\mathrm{MPSI}}$ *with statistical security in the* $\mathcal{F}_{OT}$*-hybrid model,*[5] *in the* augmented *semi-honest model, for the right choice of parameters* m *and* k *as functions of the security parameter* κ *and the bound* M *on the size of each individual data set.*

The composition theorem of [4], immediately yields the following corollary.

Corollary 2. *Assume trap-door permutations exist. Then, protocol* MPSI-*Aug securely computes* $\mathcal{F}_{\mathrm{MPSI}}$ *in the* augmented *semi-honest model, for the right choice of parameters* m *and* k *as functions of the security parameter* κ *and the bound* M *on the size of each individual data set.*

4 Implementations and Experimental Results

We implemented all three versions of our protocol with an emphasis on the augmented semi-honest construction, as we find it more comparable to previous implementations of [6] and of [18]. Our implementations are based on the open source code of [6]. Nevertheless, we incorporated several changes and optimizations, and generalized the implementation from the two-party setting.

We ran our experiments on a cluster with a very low latency network called CREATE [22] (which is part of the DETER project). The cluster is comprised of Intel XEON 2.20 GHz machines (E5-2420) with 6 cores running Linux (Ubuntu 16.04 x86-64), and the ping time between computers is approximately 0.1ms and 1Gb of symmetric bandwidth. We survey the results of each of the variants in a separate table, surveying the effect of the main optimizations incorporated.

See Table 4 for the experimental results of the implementation of the augmented semi-honest protocol – ran in a *h*igh latency network, and Table 3 for the experimental results of that protocol ran over a low latency network.

Code. Our code is written in Java, using OpenJDK Runtime Environment (version 1.8.0). We view this choice as a first step, which was easier given the implementation of [6]. We believe that translating our code into a C++ implementation would result in a factor of two improvement to its running time. We leave this as future work.

Augmented Semi-Honest Protocol

Input: All parties in the set $\mathcal{P} = \{\mathsf{P}_0, \ldots, \mathsf{P}_{t-1}\}$ hold the security parameter κ, the number of parties t, and the parameters m, k, and λ – for the Bloom filter length, the number of hash functions, and the length of each string in the GBF, respectively. In addition, each party P_i holds its private data set $\mathcal{DB}_i$.

Initialization: P_0 randomly selects a sequence $H \subseteq \mathcal{H}$ of k hash functions, and sends its description (i.e., keys) to all other parties.

Local GBFs generation:
Party P_0 creates a local $\mathrm{BF}_{\mathcal{DB}_0} = \mathsf{BuildBF}(\mathcal{DB}_0, k, m, H)$.
Each $\mathsf{P}_i \in \mathcal{P} \setminus \mathsf{P}_0$ acts as follows:

1. Creates a local $\mathrm{GBF}_{\mathcal{DB}_i} = \mathsf{BuildGBF}(\mathcal{DB}_i, k, m, H, \lambda)$.
2. Selects uniform strings (shares) $\left\{\mathrm{GBF}^j_{\mathcal{DB}_i}\right\}_{j=1}^{t-1}$ conditioned on $\bigoplus_{j=1}^{t-1} \mathrm{GBF}^j_{\mathcal{DB}_i} = \mathrm{GBF}_{\mathcal{DB}_i}$ (i.e., selects shares in a $(t-1)$-out-of-$(t-1)$ scheme).

Obtaining intermediate shares: Each $\mathsf{P}_i \in \mathcal{P} \setminus \mathsf{P}_0$ sends to each party $\mathsf{P}_j \in \mathcal{P} \setminus \mathsf{P}_0$ the share $\mathrm{GBF}^j_{\mathcal{DB}_i}$. Party P_j in turn constructs its share of the (partial) intersection GBF by computing
$\mathrm{GBF}^j_{\mathcal{DB}_{\mathcal{IS}'}} = \bigoplus_{i=1}^{t-1} \mathrm{GBF}^j_{\mathcal{DB}_i}$.

***OT* interactions:** [a]
The server, party P_0 (taking the role of the receiver) engages with each $\mathsf{P}_j \in \mathcal{P} \setminus \mathsf{P}_0$ (taking the role of the sender) in an OT interaction for each $\ell \in [m]$. Party P_j uses $s_0^{(j,\ell)}, s_1^{(j,\ell)}$ as its input strings to the OT, where $s_0^{(j,\ell)} \leftarrow \{0,1\}^\lambda$ and $s_1^{(j,\ell)} = \mathrm{GBF}[j][\ell]$. Party P_0 uses $\mathrm{BF}_{\mathcal{DB}_0}[\ell]$ as input bits to the OT.
Denote by ${\mathrm{GBF}^{\star}}^0_{\mathcal{DB}_j}$ the vector whose entry ${\mathrm{GBF}^{\star}}^0_{\mathcal{DB}_j}[\ell]$ is the output of party P_0 in the ℓ'th OT interaction with P_j.

Output reconstruction by P_0: P_0 computes $\mathrm{GBF}_{\mathcal{IS}} = \bigoplus_{j=1}^{t-1} {\mathrm{GBF}^{\star}}^0_{\mathcal{DB}_j}$, and $\mathcal{IS} = \{x \in \mathcal{DB}_0 \colon \mathrm{GBFQuery}(\mathrm{GBF}_{\mathcal{IS}}, \mathit{item}) = \mathrm{True}\}$.

Output: P_0 outputs $\mathcal{IS}$.

[a] Here we describe the basic protocol, without the hypercube routing optimization. The optimized version is obtained by replacing the OT interaction step with the one described in Section 4.1.

Fig. 2: Protocol *MPSI*-Aug – multiparty private set intersection with augmented semi-honest security

4.1 Optimizing Communication via Hypercube Routing

The most significant optimization we have incorporated is in the communication scheme of the protocol, which is now performed over a hypercube spanning tree. Recall that in the star protocol the server P_0 engages in an OT interaction with all other parties. In order to reduce the overall latency, we wish to load balance the interactions between pairs of parties. Hypercube routing was originally used to speed up message propagation by replacing a star-like propagation scheme with a tree like scheme [2]. In each step, all parties that have already received the message forward it to its destination via their neighbors. To transmit the shares to the servers, we use the reverse order of communication. In addition, rather than just sending a message, the full OT interaction takes place.

Assume that the number of parties is $t = 2^\ell$. Let e_j be a binary vector of length ℓ, in which the bit in location j is set to 1 and all other bits are set to 0. In the hypercube scheme, at time $0 \leq j \leq \ell - 1$, each party i whose identity has 0 in all bits $0, \ldots, j$, runs the OT protocol with party $i \oplus e_j$, where party i is the receiver. It is straightforward to see that P_0 (the server) is the receiver in all rounds, and that at the end of the protocol it learns information that it is indistinguishable from the information it learns in the star protocol.

The number of interactions run by P_0 is reduced from being equal to the number of parties t to being $\ell = \log t$. In the CREATE environment that we used, the original protocol, without the hypercube optimization, could not exceed 524288 items per data set and 12 parties, or otherwise it would crash (see Figure 3). However, we may expect to be able to run the hypercube based protocol with as many as $2^{12} = 4096$ parties with the same dataset size.

Nevertheless, our experiments demonstrated that if the flow of information is done round by round by all parties (as specified by the hypercube method), the running time is much slower than one would expect. We observe that allowing the parties to start interacting with parties for 'future' rounds, before completing the interaction for the current round (with another party) proves highly beneficial. In this manner, the order of communication is no longer predefined, however, this flexibility turns out to give the protocol's running time a great boost. The effect of this additional optimization, referred to as the *n*o-blocking hypercube is illustrated in Table 3. It should be noted that this optimization balances the load not only in terms of RAM resources, but also in bandwidth and CPU load.

Remark 1 (proving the security of the hypercube communication optimization). We stress that the proof of the augmented semi-honest protocol with the hypercube optimization goes through, similarly to the original semi-honest protocol. One change that is required in the protocol is to have the server participate in the creation and sharing of the intersection GBF (with all other parties) – before the OT phase starts. Intuitively, this deals with the case that the server is not corrupted, and all honest parties engage in OT interaction with a subset of the corrupted parties.

4.2 Optimizing the Computation

Using Murmur3[14]/xxHash[15] in the Bloom filter. The implementation of the garbled Bloom filter in [6] uses SHA-1 to map values to locations in the filter. Since there is no need to use a cryptographic hash function for this purpose, we replaced SHA-1 with the non-cryptographic hash functions Murmur3 and xxHash (which are commonly used in algorithms). This turned out to substantially improve the run time of the GBF creation.

Cached memory misses optimization. The implementation of [6] used a two dimensional array to hold the GBFs. In our implementation, which requires many XOR operations, this resulted in many cache misses. We changed the implementation to store the GBFs in a cache-aware manner. The effect of this optimization was more evident as the data set size (and, hence, share size) grew. For example, for 524288-bit long shares, this optimization shaved off up to 60% of the time it took to construct share in the two dimensional array of [6].

Local share reconstruction – parallel computation. The local computations of the XOR operations can be improved by running them in parallel by multiple threads. However, this requires a substantial part of the RAM to be occupied at all time. to reduce RAM usage we break the bit-strings into blocks and compute the XOR block-wise and in parallel. In some more detail, we create two PRG threads for each party P_i, one for handling the shares (seeds) P_i sends to other parties, and the other for the shares it receives. Both threads run in parallel and divide the shares into blocks of a predefined length. After creating the blocks, the XOR is applied block-wise to all shares in parallel, independently of each other. Because the XOR operation is faster than the PRG operation, blocks of the shares are removed shortly after their creation, leaving enough memory free and usable for upcoming block XOR computations. The improvements of this optimization, as well as the previous one, are illustrated in Table 5.

Sending short seeds instead of full payload. The parties share their local GBF with each other. To improve the communication complexity, rather than sending the full random sharing to each other, parties send a short seed such that the receiving party can expand this seed and calculate its final share. payload locally. We stress that, since all parties hold the same key, in order to claim security of our implementations, we need to model the PRG or the PRF as a We further stress that secret sharing only takes place during offline preprocessing, and hence, not incorporating this optimization does not affect the online time of our constructions.

Augmented Semi-Honest Star Communication	**Parties**	**1024**	**65536**	**131072**	**262144**	**524288**
	2	2.25	12.99	28.23	54.81	149.21
	3	1.97	32.01	64.83	85.80	229.80
	4	2.08	25.03	47.56	91.48	268.56
	6	2.36	23.84	63.89	110.08	238.69
	12	2.29	39.22	67.87	195.51	493.21
	18	4.17	45.49	108.11	276.14	-
	24	5.48	59.97	132.48	339.10	-
	32	6.10	83.21	190.47	-	-
	36	6.62	102.82	244.15	-	-
Augmented Semi-Honest Hyper Cube Communication	**2**	2.00	11.46	28.82	63.67	152.99
	3	3.49	27.91	43.34	136.57	324.12
	4	3.00	30.21	56.09	205.00	437.25
	6	4.52	34.15	72.90	184.00	376.14
	12	5.31	45.42	87.05	206.00	430.98
	18	6.38	46.00	98.80	219.36	507.29
	24	6.43	52.00	101.07	215.38	565.17
	32	5.66	58.67	114.57	234.11	581.36
	36	7.30	63.46	125.34	237.53	652.46
Augmented Semi-Honest Hyper Cube Communication, No Blocking.	**2**	1.98	11.97	31.09	79.28	172.00
	3	1.95	32.11	62.85	108.82	224.21
	4	2.49	33.54	88.18	142.65	341.40
	6	2.58	30.10	78.40	149.23	267.56
	12	3.07	38.53	71.69	239.00	384.04
	18	3.44	43.00	90.56	204.94	486.11
	24	3.57	46.04	101.64	203.03	499.00
	32	3.82	51.85	112.21	218.69	418.00
	36	3.87	61.93	107.57	252.06	557.42
	56	4.06	62.92	136.58	294.37	744.29

Table 3: Time measurements (in seconds) – augmented semi-honest protocol – low latency. Rows indicate number of parties. Colomns indicate data set size. Results appear for (i) basic star topology (no hypercube optimization), (ii) basic hypercube optimization (allows blocking), and (iii) hypercube optimization without blocking.

		Items				
		1024	**65536**	**131072**	**262144**	**524288**
Parties	**2**	3.91	32.78	66.85	142.08	307.52
	3	3.93	70.066	139.2	243.44	512.91
	4	6.64	75.09	142.48	311.46	612.76
	6	6.88	90.39	174.41	360.32	825.24
	12	9.07	117.67	227.19	467.78	1045.63
	18	11.46	140.55	266.23	536.11	1181.91
	24	12.01	142.9	295.96	594.37	1250.43
	32	13.76	160.15	310.02	628.95	1342.29
	36	13.8	167.34	326.73	666.91	1469.35

Table 4: Time measurements – *MPSI*-Aug protocol – **50ms latency, 100Mb bandwidth**. Time in seconds.

		Items				
	Shares	**1024**	**65536**	**131072**	**262144**	**524288**
Two-Dimensional Array GBF of [6]	**1**	0.08	7.33	15.71	29.02	76.31
	2	0.17	12.53	26.17	52.58	142.26
	4	0.32	21.81	50.44	92.00	256.45
	8	0.66	37.47	76.71	169.87	407.49
	12	0.86	53.72	111.32	229.89	672.39
	24	1.69	101.17	207.84	446.05	1261.62
	36	2.45	151.35	310.74	641.65	1813.56
	64	4.23	268.46	534.31	1111.70	3092.48
One-Dimensional Array	**1**	0.07	3.81	7.17	14.49	28.80
	2	0.13	7.42	14.40	28.91	57.27
	4	0.29	14.74	29.02	57.97	116.68
	8	0.58	29.26	58.08	115.46	248.23
	12	0.85	43.40	88.11	181.43	368.56
	24	1.47	87.52	172.45	350.52	685.31
	36	2.15	128.71	255.90	519.93	1059.23
	64	3.81	229.66	459.14	906.53	1852.85
One-Dimensional Array with Parallel Computation of XOR Operations	**1**	0.11	5.97	9.70	30.79	63.95
	2	0.10	5.65	12.15	34.38	64.52
	4	0.13	4.36	8.62	43.92	51.45
	8	0.16	7.32	15.89	39.92	37.47
	12	0.20	10.05	14.55	31.74	113.77
	24	0.35	17.85	46.70	92.09	181.84
	36	0.37	23.95	38.66	100.75	208.85
	64	0.58	24.05	62.84	85.88	199.61

Table 5: Time Measurement (in seconds) of Share Creation ($\kappa = 80$).

Bibliography

[1] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1–10, 1988.

[2] D. P. Bertsekas, C. Özveren, G. D. Stamoulis, P. Tseng, and J. N. Tsitsiklis. Optimal communication algorithms for hypercubes. *Journal of Parallel and Distributed Computing*, 11(4):263–275, 1991.

[3] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[4] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY*, 13(1):143–202, 2000.

[5] E. D. Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In R. Sion, editor, *14th International Financial Cryptography and Data Security Conference*, volume 6052 of *Lecture Notes in Computer Science*, pages 143–159. Springer, 2010.

[6] C. Dong, L. Chen, and Z. Wen. When private set intersection meets big data: An efficient and scalable protocol. In *the ACM Conference on Computer and Communications Security, CCS'13*, pages 789–800. ACM, 2013.

[7] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2004.

[8] M. J. Freedman, C. Hazay, K. Nissim, and B. Pinkas. Efficient set intersection with simulation-based security. *J. Cryptology*, 29(1):115–155, 2016.

[9] O. Goldreich. *Foundations of Cryptography – Volume 2: Basic Applications*. Cambridge University Press, 2004.

[10] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *stoc19*, pages 218–229, 1987.

[11] C. Hazay and M. Venkitasubramaniam. Scalable multi-party private set-intersection. In *IACR International Workshop on Public Key Cryptography*, pages 175–203. Springer, 2017.

[12] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In D. Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 145 – 161. Springer, 2003.

[13] S. Jarecki and X. Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In O. Reingold, editor, *Theory of Cryptography, Sixth Theory of Cryptography Conference, TCC 2009*, pages 577–594, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[14] P. Jayachandran. Murmur hash algorithm. `https://github.com/prasanthj/hasher`, 2014. [Online; accessed 6-October-2017].

[15] P. Jayachandran. xxHash hash algorithm. `https://github.com/prasanthj/hasher`, 2014. [Online; accessed 6-October-2017].

[16] V. Kolesnikov and R. Kumaresan. Improved OT extension for transferring short secrets. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013*, volume 8042 of *Lecture Notes in Computer Science*, pages 54–70. Springer, 2013.

[17] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 818–829, 2016.

[18] V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek, and N. Trieu. Practical multiparty private set intersection from symmetric-key techniques. In *the ACM Conference on Computer and Communications Security, CCS'17*, 2017.

[19] B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on OT extension. In *Proceedings of the 23rd USENIX Security Symposium*, pages 797–812. USENIX Association, 2014.

[20] B. Pinkas, T. Schneider, and M. Zohner. Scalable private set intersection based on OT extension. Cryptology ePrint Archive, Report 2016/930, 2016.

[21] P. Rindal and M. Rosulek. Faster malicious 2-party secure computation with online/offline dual execution. In *25th USENIX Security Symposium, USENIX Security*, pages 297–314. USENIX Association, 2016.

[22] I.-U. C. C. I. C. Unit. Cyber Research, Experimentation and Test Environment. https://createlab.iucc.ac.il/, 2017. [Online; accessed 16-October-2017].

[23] A. C. Yao. Protocols for secure computations. In *Proceedings of the 23th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 160–164, 1982.