Improved Private Set Intersection against Malicious Adversaries

Peter Rindal* Mike Rosulek*

July 30, 2016

Abstract

Private set intersection (PSI) refers to a special case of secure two-party computation in which the parties each have a set of items and compute the intersection of these sets without revealing any additional information. In this paper we present improvements to practical PSI providing security in the presence of *malicious* adversaries.

Our starting point is the protocol of Dong, Chen & Wen (CCS 2013) that is based on Bloom filters. We identify a bug in their malicious-secure variant and show how to fix it using a cut-and-choose approach that has low overhead while simultaneously avoiding one the main computational bottleneck in their original protocol. We also point out some subtleties that arise when using Bloom filters in malicious-secure cryptographic protocols.

We have implemented our PSI protocols and report on its performance. Our improvements reduce the cost of Dong et al.'s protocol by a factor of $8-75\times$ on a single thread. For instance, our protocol has an online time of 14 seconds and an overall time of 3.3 minutes to securely compute the intersection of two sets of 1 million items each.

1 Introduction

Private set intersection (PSI) is a cryptographic primitive that allows two parties holding sets X and Y, respectively, to learn the intersection $X \cap Y$ while not revealing any additional information about X and Y.

PSI has a wide range of applications: contact discovery [Mar14], secret handshakes [HFH99], measuring advertisement conversion rates, and securely sharing security incident information [PSSZ15], to name a few.

There has been a great deal of recent progress in efficient PSI protocols that are secure against *semi-honest* adversaries, who are assumed to follow the protocol. The current state of the art has culminated in extremely fast PSI protocols. The fastest one, due to Pinkas et al. [PSSZ15], can securely compute the intersection of two sets, each with 2^{20} items of 32 bits, in approximately 4 seconds.

Looking more closely, the most efficient semi-honest protocols are those that are based on **oblivious transfer (OT) extension.** Oblivious transfer is a fundamental cryptographic primitive (see Figure 1). While in general OT requires expensive public-key computations, the idea of OT extension [Bea96, IKNP03] allows the parties to efficiently realize any number of *effective* OTs by using only a small number (e.g., 128) of *base OTs* plus some much more efficient symmetric-key computations. Using OT extension, oblivious transfers become extremely inexpensive in practice. Pinkas et al. [PSZ14] compared many paradigms for PSI and found the ones based on OTs are much more efficient than those based on algebraic & public-key techniques.

Our contributions In many settings, security against semi-honest adversaries is insufficient. *Our goal in this paper is to translate the recent success in semi-honest PSI to the setting of* ***malicious security.*** Following the discussion above, this means focusing on PSI techniques based on oblivious transfers. Indeed, recent protocols for OT extension against *malicious* adversaries [ALSZ13, KOS15] are almost as efficient as (only a few percent more expensive than) OT extension for semi-honest adversaries.

*Oregon State University, {rindalp,rosulekm}@eecs.oregonstate.edu. Supported by NSF award 1149647 and a Google Research Award. The first author is also supported by an ARCS foundation fellowship.

Our starting point is the protocol paradigm of Dong, Chen & Wen [DCW13] that is based on OTs and Bloom filter encodings. We describe their approach in more detail in Section 3. In their work they describe one of the few malicious-secure PSI protocols based primarily on OTs rather than algebraic public-key techniques. We present the following improvements and additions to their protocol:

1. Most importantly, we show that their protocol has a subtle security flaw, which allows a malicious sender to induce inconsistent outputs for the receiver. We present a fix for this flaw, using a very lightweight cut-and-choose technique.

2. We present a full simulation-based security proof of the Bloom-filter-based PSI paradigm. In doing so, we identify a subtle but important aspect about using Bloom filters in a protocol meant to provide security in the presence of malicious adversaries. Namely, the simulator must be able to extract all items stored in an adversarially constructed Bloom filter. We argue that this capability is an *inherently* non-standard model assumption, in the sense that it seems to require the Bloom filter hash functions to be modeled as (non-programmable) random oracles. Details are in Section 5.1.

3. We implement both the original DCW protocol and our improved version. We find that the major bottleneck in the original DCW protocol is not in the cryptographic operations, but actually in a polynomial interpolation computation. The absence of polynomial interpolation in our new protocol (along with our other improvements) decreases the running time by a factor of over 8-75x.

1.1 Related Work

As mentioned above, our work builds heavily on the protocol paradigm of Dong et al. [DCW13] that uses Bloom filters and OTs. We discuss this protocol in great detail in Section 3.

Several other paradigms for PSI have been proposed. Currently the fastest protocols in the *semi-honest* setting are those of Pinkas et al. [PSZ14, PSSZ15] that rely heavily on oblivious transfers. Adapting these protocols to the malicious setting is highly non-trivial, and we were unsuccessful in doing so.

Here we list other protocol paradigms that allow for malicious security when possible. The earliest technique for PSI is the elegant Diffie-Hellman-based protocol of [HFH99]. Protocols in this paradigm achieving security against malicious adversaries include [DKT10].

Freedman et al. [FNP04] describe a PSI paradigm based on oblivious polynomial evaluation, which was extended to the malicious setting in [DMRY09].

Huang et al. [HEK12] explored using general-purpose 2PC techniques (e.g., garbled circuits) for PSI. Several improvements to this paradigm were suggested in [PSSZ15]. Malicious security can be achieved in this paradigm in a generic way, using any cut-and-choose approach, e.g., [Lin13].

Kamara et al. [KMRS14] presented PSI protocols that take advantage of a semi-trusted server to achieve extremely high performance. Our work focuses on the more traditional setting with just 2 parties.

2 Preliminaries

We use κ to denote a computational security parameter (e.g., $\kappa = 128$ in our implementations), and λ to denote a statistical security parameter (e.g., $\lambda = 40$ in our implementations). We use $[n]$ to denote the set $\{1, \dots, n\}$.

2.1 Efficient Oblivious Transfer

Our protocol makes use of 1-out-of-2 oblivious transfer (OT) of random messages. The ideal functionality is described in Figure 1. We require a large number of such OTs, secure against malicious adversaries. These can be obtained efficiently via OT extension [Bea96]. The idea is to perform a fixed number (e.g., 128) of "base OTs", and from this correlated randomness derive a large number of effective OTs using only symmetric-key primitives.

The two most efficient OT extension protocols providing malicious security are those of [ALSZ15, KOS15], which are based on the semi-honest secure paradigm of [IKNP03].

Parameters: ℓ is the length of the OT strings.

- On input $b \in \{0,1\}$ from the receiver, sample messages $m_0, m_1 \leftarrow \{0,1\}^\ell$. Give output m_b to the receiver and give (m_0, m_1) to the sender.

Figure 1: Ideal functionality for 1-out-of-2 OT of random messages

Parameters: σ is the bit-length of the parties' items. n is the size of the honest parties' sets. $n' > n$ is the allowed size of the corrupt party's set.

- On input $Y \subseteq \{0,1\}^\sigma$ from Bob, ensure that $|Y| \leq n$ if Bob is honest, and that $|Y| \leq n'$ if Bob is corrupt. Give output BOB-INPUT to Alice.
- Thereafter, on input $X \subseteq \{0,1\}^\sigma$ from Alice, likewise ensure that $|X| \leq n$ if Alice is honest, and that $|X| \leq n'$ if Alice is corrupt. Give output $X \cap Y$ to Bob.

Figure 2: Ideal functionality for private set intersection (with one-sided output)

2.2 Private Set Intersection

In Figure 2 we give the ideal functionality that specifies the goal of private set intersection. We point out several facts of interest. (1) The functionality gives output only to Bob. (2) The functionality allows corrupt parties to provide larger input sets than the honest parties. This reflects that our protocol is unable to strictly enforce the size of an adversary's set to be the same as that of the honest party. We elaborate when discussing the security of the protocol.

We define security of a PSI protocol using the standard paradigm of 2PC. In particular, our protocol is secure in the *universal composability (UC)* framework of Canetti [Can01]. Security is defined using the real/ideal, simulation-based paradigm that considers two interactions:

- In the **real interaction**, a malicious adversary $\mathcal{A}$ attacks an honest party who is running the protocol π. The honest party's inputs are chosen by an *environment* $\mathcal{Z}$; the honest party also sends its final protocol output to $\mathcal{Z}$. The environment also interacts arbitrarily with the adversary. Our protocols are in a *hybrid* world, in which the protocol participants have access to an ideal random-OT functionality (Figure 1). We define $\textsc{real}[\pi, \mathcal{Z}, \mathcal{A}]$ to be the (random variable) output of $\mathcal{Z}$ in this interaction.
- In the **ideal interaction**, a malicious adversary $\mathcal{S}$ and an honest party simply interact with the ideal functionality $\mathcal{F}$ (in our case, the ideal PSI protocol of Figure 2). The honest party simply forwards its input from the environment to $\mathcal{F}$ and its output from $\mathcal{F}$ to the environment. We define $\textsc{ideal}[\mathcal{F}, \mathcal{Z}, \mathcal{S}]$ to be the output of $\mathcal{Z}$ in this interaction.

We say that a protocol π **UC-securely realizes** functionality $\mathcal{F}$ if: for all PPT adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$, such that for all PPT environments $\mathcal{Z}$:

$$\textsc{real}[\pi, \mathcal{Z}, \mathcal{A}] \approx \textsc{ideal}[\mathcal{F}, \mathcal{Z}, \mathcal{S}]$$

where "$\approx$" denotes computational indistinguishability.

Our protocol uses a (non-programmable) random oracle. In Section 5.4 we discuss technicalities that arise when modeling such global objects in the UC framework.

2.3 Bloom Filters

A **Bloom filter (BF)** is an N-bit array B associated with k random functions $h_1, \ldots, h_k : \{0,1\}^* \to [N]$. To store an item x in the Bloom filter, one sets $B[h_i(x)] = 1$ for all i. To check the presence of an item x in the Bloom filter, one simply checks whether $B[h_i(x)] = 1$ for all i. Any item stored in the Bloom filter will therefore be detected when queried; however, *false positives* are possible.

3 The DCW Protocol Paradigm

The PSI protocol of Dong, Chen, and Wen [DCW13] (hereafter DCW) is based on representing the parties' input sets as Bloom filters (BFs). We describe the details of their protocol in this section.

If B and B' are BFs for two sets S and S', using the same parameters (including the same random functions), then it is true that $B \wedge B'$ (bit-wise AND) is a BF for $S \cap S'$. However, one cannot construct a PSI protocol simply by computing a bit-wise AND of Bloom filters. The reason is that $B \wedge B'$ leaks more about S and S' than their intersection $S \cap S'$. For example, consider the case where $S \cap S' = \emptyset$. Then the most natural Bloom filter for $S \cap S'$ is an all-zeroes string, and yet $B \wedge B'$ may contain a few 1s with noticeable probability. The location of these 1s depends on the items in S and S', and hence cannot be simulated just by knowing that $S \cap S' = \emptyset$.

DCW proposed a variant Bloom filter that they call a **garbled Bloom filter** (GBF). In a GBF G meant to store m-bit strings, each $G[i]$ is itself an m-bit string rather than a single bit. Then an item x is stored in G by ensuring that $x = \bigoplus_i G[h_i(x)]$. That is, the positions indexed by hashing x should store additive secret shares of x. All other positions in G are chosen uniformly.

The **semi-honest** PSI protocol of DCW uses GBFs in the following way. The two parties agree on Bloom filter parameters. Alice prepares a GBF G representing her input set. The receiver Bob prepares a standard BF B representing his input set. For each position i in the Bloom filters, the parties use oblivious transfer so that Bob can learn $G[i]$ (a string) iff $B[i] = 1$. These are exactly the positions of G that Bob needs to probe in order to determine which of his inputs is stored in G. Hence Bob can learn the intersection. DCW prove that this protocol is secure. That is, they show that Bob's view $\{G[i] \mid B[i] = 1\}$ can be simulated given only the intersection of Alice and Bob's sets.

DCW also describe a **malicious-secure** variant of their GBF-based protocol. The main challenge is that nothing in the semi-honest protocol prevents a malicious Bob from learning *all* of Alice's GBF G. This would reveal Alice's entire input, which can only be simulated in the ideal world by Bob sending the entire universe $\{0,1\}^\sigma$ as input. Since in general the universe is exponentially large, this behavior is unsimulatable and hence constitutes an attack.

To prevent this, DCW propose to use 1-out-of-2 OTs in the following way. Bob can choose to either pick up a position $G[i]$ in Alice's GBF (if Bob has a 1 in $B[i]$) or else learn a value s_i (if Bob has a 0 in $B[i]$). The values s_i are an $N/2$-out-of-N secret sharing of some secret s^* which is used to encrypt all of the $G[i]$ values. Hence, Alice's inputs to the ith OT are $(s_i, \mathsf{Enc}(s^*, G[i]))$, where Enc is a suitable encryption scheme. Intuitively, if Bob tries to obtain too many positions of Alice's GBF (more than half), then he cannot recover the key s^* used to decrypt them.

As long as $N > 2k|Y|$ (where Y is Bob's input set), an honest Bob is guaranteed to have at least half of his BF bits set to zero. Hence, he can reconstruct s^* from the s_i shares, decrypt the $G[i]$ values, and probe these GBF positions to learn the intersection. We describe the protocol formally in Figure 3.

Parameters: X is Alice's input, Y is Bob's input. N is the required Bloom filter size; We assume the parties have agreed on common BF parameters.

1. Alice chooses a random key $s^* \in \{0,1\}^\kappa$ and generates an $N/2$-out-of-N secret sharing $(s_1, \ldots, s_N)$.

2. Alice generates a GBF G encoding her inputs X. Bob generates a standard BF B encoding his inputs Y.

3. For $i \in [N]$, the parties invoke an instance of 1-out-of-2 OT, where Alice gives inputs $(s_i, c_i = \mathsf{Enc}(s^*, G[i]))$ and Bob uses choice bit $B[i]$.

4. Bob reconstructs s^* from the set of shares $\{s_i \mid B[i] = 0\}$ he obtained in the previous step. Then he uses s^* to decrypt the ciphertexts $\{c_i \mid B[i] = 1\}$, obtaining $\{G[i] \mid B[i] = 1\}$. Finally, he outputs $\{y \in Y \mid y = \bigoplus_i G[h_i(y)]\}$.

Figure 3: The malicious-secure protocol of DCW [DCW13].

3.1 Insecurity of the DCW Protocol

Unfortunately, the malicious-secure variant of DCW is not secure![1] We now describe an a attack on their protocol, which was independently & concurrently discovered by Lambæk [Lam16]. A corrupt Alice will generate s_i values that are *not* a valid $N/2$-out-of-N secret sharing. DCW do not specify Bob's behavior when obtaining invalid shares. However, we argue that no matter what Bob's behavior is (e.g., to abort in this case), Alice can violate the security requirement.

As a concrete attack, let Alice honestly generate shares s_i of s^*, but then change the value of s_1 in any way. She otherwise runs the protocol as instructed. If the first bit of Bob's Bloom filter is 1, then this deviation from the protocol is invisible to him, and Alice's behavior is indistinguishable from honest behavior. Otherwise, Bob will pick up s_1 which is not a valid share. If Bob aborts in this case, then his abort probability depends on whether his first BF bit is 1. The effect of this attack on Bob's output cannot be simulated in the ideal PSI functionality, so it represents a violation of security.

Even if we modify Bob's behavior to gracefully handle some limited number of invalid shares, there must be some threshold of invalid shares above which Bob (information theoretically) cannot recover the secret s^*. Whether or not Bob recovers s^* therefore depends on *individual bits* of his Bloom filter. And whether we make Bob abort or do something else (like output $\emptyset$) in the case of invalid shares, the result cannot be simulated in the ideal world. Lambæk [Lam16] points out further attacks, in which Alice can cleverly craft shares and encryptions of GBF values to cause her effective input to depend on Bob's inputs (hence violating input independence).

4 Our Protocol

The spirit of DCW's malicious protocol is to restrict the adversary from setting too many 1s in its Bloom filter, thereby learning too many positions in Alice's GBF. In this section, we show how to achieve the spirit of the DCW protocol using a lightweight cut-and-choose approach.

The high-level idea is to generate slightly more 1-out-of-2 OTs than the number of BF bits needed. Bob is supposed to use a limited number of 1s for his choice bits. To check this, Alice picks a small random fraction of the OTs and asks Bob to prove that an appropriate number of them used choice bit 0. If Alice uses *random* strings as her choice-bit-0 messages, then Bob can prove his choice bit by simply reporting this string.[2] If Bob cannot prove that he used sufficiently many 0s as choice bits, then Alice aborts. Otherwise, Alice has high certainty that the unopened OTs contain a limited number of choice bits 1.

After this cut-and-choose, Bob can choose a permutation that reorders the unopened OTs into his desired BF. In other words, if $c_1, \ldots, c_N$ are Bob's choice bits in the unopened OTs, Bob sends a random π such that $c_{\pi(1)}, \ldots, c_{\pi(N)}$ are the bits of his desired BF. Then Alice can send her GBF, masked by the choice-bit-1 OT messages permuted in this way.

We discuss the required parameters for the cut-and-choose below. However, we remark that the overhead is minimal. It increases the number of required OTs by only 1–10%.

4.1 Additional Optimizations

Starting from the basic outline just described, we also include several important optimizations. The complete protocol is described formally in Figure 4.

Random GBF In their treatment of the *semi-honest* DCW protocol, Pinkas et al. [PSZ14] suggested an optimization that eliminates the need for Alice to send her entire masked GBF. Suppose the parties use 1-out-of-2 OT of *random* messages (i.e., the sender Alice does not choose the OT messages; instead, they are chosen randomly by the protocol / ideal functionality). In this case, the concrete cost of OT extension is greatly reduced (cf. [ALSZ13]). Rather than generating a GBF of her inputs, Alice generates an array G where $G[i]$ is the random OT message in the ith OT corresponding to bit 1 (an honest Bob learns $G[i]$ iff the ith bit of his Bloom filter is 1).

[1] We contacted the authors of [DCW13], who confirmed that our attack violates malicious security.
[2] This *committing* property of an OT choice bit was pointed out by Rivest [Riv99].

Rather than arranging for $\bigoplus_i G[h_i(x)] = x$, as in a garbled BF, the idea is to let the G-values be random and have Alice directly send to Bob a **summary value** $K_x = \bigoplus_i G[h_i(x)]$ for each of her elements x. For each item y in Bob's input set, he can likewise compute K_y since he learned the values of G corresponding to 1s in his Bloom filter. Bob can check to see whether K_y is in the list of strings sent by Alice. For items x not stored in Bob's Bloom filter, the value K_x is random from his point of view.

Pinkas et al. show that this optimization significantly reduces the cost, since most OT extension protocols require less communication for OT of random messages. In particular, Alice's main communication now depends on the number of items in her set rather than the size of the GBF encoding her set. Although the optimization was suggested for the semi-honest variant of DCW, we point out that it also applies to the malicious variant of DCW and to our cut-and-choose protocol.

In the malicious-secure DCW protocol, the idea is to prevent Bob from seeing GBF entries unless he has enough shares to recover the key s^*. To achieve the same effect with a random-GBF, we let the choice-bit-1 OT messages be random (choice-bit-0 messages still need to be chosen messages: secret shares of s^*). These choice-bit-1 OT messages define a random GBF G for Alice. Then instead of sending a summary value $\bigoplus_i G[h_i(x)]$ for each x, Alice sends $[\bigoplus_i G[h_i(x)]] \oplus F(s^*, x)$, where F is a pseudorandom function. If Bob does not use choice-bit-0 enough, he does not learn s^* and all of these messages from Alice are pseudorandom.

In our protocol, we can let both OT messages be random, which significantly reduces the concrete overhead. The choice-bit-0 messages are used when Bob proves his choice bit in the cut-and-choose step. The choice-bit-1 messages are used as a random GBF G, and Alice sends summary values just as in the semi-honest variant.

We also point out that Pinkas et al. and DCW overlook a subtlety in how the summary values and the GBF should be constructed. Pinkas et al. specify the summary value as $\bigoplus_i G[h_i(x)]$ where h_i are the BF hash functions. With noticeable probability there may be a collision under two hash functions for the same x — that is, $h_i(x) = h_{i'}(x)$. If this happens, the term $G[h_i(x)] = G[h_{i'}(x)]$ can cancel itself out from the XOR summation and the summary value will not depend on this term. The DCW protocol also has an analogous issue.[3] As such, Bob has a better chance of guessing this summary value, which leads to a potential weakness in security. We fix this by computing the summary value using an XOR expression that eliminates the problem of colliding terms:

$$\bigoplus_{j \in h_*(x)} G[j], \qquad \text{where } h_*(x) \stackrel{\text{def}}{=} \{h_i(x) : i \in [k]\}.$$

Note that in the event of a collision among BF hash functions, we get $|h_*(x)| < k$.

Finally, for technical reasons, it turns out to be convenient in our protocol to define the summary value of x to be $H(x \| \bigoplus_{j \in h_*(x)} G[j])$ where H is a (non-programmable) random oracle.[4]

Hash only "on demand." In OT-extension for random messages, the parties compute the protocol outputs by taking a hash of certain values derived from the base OTs. Apart from the base OTs (whose cost is constant), these hashes account for essentially all the cryptographic operations in our protocol. We therefore modify our implementation of OT extension so that these hashes are not performed until the values are needed. In our protocol, only a small number (e.g., 1%) of the choice-bit-0 OT messages are ever used (for the cut-and-choose check), and only about half of the choice-bit-1 OT messages are needed by the sender (only the positions that would be 1 in a BF for the sender's input). Hence, the reduction in cost for the receiver is roughly 50%, and the reduction for the sender is roughly 75%. A similar optimization was also suggested by Pinkas et al. [PSZ14], since the choice-bit 0 messages are not used at all in the semi-honest protocol.

Aggregating proofs-of-choice-bits Finally, we can reduce the communication cost of the cut-and-choose step. Recall that Bob must prove that he used choice bit 0 in a sufficient number of OTs. For the ith OT, Bob can simply send $m_{i,0}$, the random output he received from the ith OT. To prove he used choice bit 0 for an entire *set* I of indices, Bob can simply send the single value $\bigoplus_{i \in I} m_{i,0}$, rather than sending each term individually.

[3] Additionally, if one strictly follows the DCW pseudocode then correctness may be violated in the event of a collision $h_i(x) = h_{i'}(x)$. If $h_i(x)$ is the first "free" GBF location then $G[h_i(x)]$ gets set to a value and then erroneously overwritten later.

[4] In practice H is instantiated with a SHA-family hash function. The XOR expression and x itself are each 128 bits.

Parameters: X is Alice's input, Y is Bob's input. N_{bf} is the required Bloom filter size; k is the number of Bloom filter hash functions; N_{ot} is the number of OTs to generate. H is modeled as a random oracle with output length κ. The choice of these parameters, as well as others $\alpha, p_{\mathsf{chk}}, N_{\mathsf{maxones}}$, is described in Section 5.2.

1. **[setup]** The parties perform a secure coin-tossing subprotocol to choose (seeds for) random Bloom filter hash functions $h_1, \dots, h_k : \{0,1\}^* \to [N_{\mathsf{bf}}]$.
2. **[random OTs]** Bob chooses a random string $b = b_1 \dots b_{N_{\mathsf{ot}}}$ with an α fraction of 1s. Parties perform N_{ot} OTs of random messages (of length κ), with Alice as sender. In the ith OT, Alice learns random strings $m_{i,0}, m_{i,1}$ chosen by the functionality. Bob uses choice bit b_i and learns $m_i^* = m_{i,b_i}$.
3. **[cut-and-choose challenge]** Alice chooses a set $C \subseteq [N_{\mathsf{ot}}]$ by choosing each index with independent probability p_{chk}. She sends C to Bob. Bob aborts if $|C| > N_{\mathsf{ot}} - N_{\mathsf{bf}}$.
4. **[cut-and-choose response]** Bob computes the set $R = \{i \in C \mid b_i = 0\}$ and sends R to Alice. To prove that he used choice bit 0 in the OTs indexed by R, Bob computes $r^* = \bigoplus_{i \in R} m_i^*$ and sends it to Alice. Alice aborts if $|C| - |R| > N_{\mathsf{maxones}}$ or if $r^* \neq \bigoplus_{i \in R} m_{i,0}$.
5. **[permute unopened OTs]** Bob generates a Bloom filter BF containing his items Y. He chooses a random injective function $\pi : [N_{\mathsf{bf}}] \to ([N_{\mathsf{ot}}] \setminus C)$ such that $BF[i] = b_{\pi(i)}$, and sends π to Alice.
6. **[randomized GBF]** For each item x in Alice's input set, she computes a summary value
$$K_x = H\left(x \,\Big\|\, \bigoplus_{i \in h_*(x)} m_{\pi(i),1} \right),$$
where $h_*(x) \stackrel{\text{def}}{=} \{h_i(x) : i \in [k]\}$. She sends a random permutation of $K = \{K_x \mid x \in X\}$.
7. **[output]** Bob outputs $\{y \in Y \mid H(y \,\|\, \bigoplus_{i \in h_*(y)} m^*_{\pi(i)}) \in K\}$.

Figure 4: Malicious-secure PSI protocol based on garbled Bloom filters.

5 Security

5.1 BF extraction

The analysis in DCW argues for malicious security in a property-based manner, but does not use a standard simulation-based notion of security. This turns out to mask a non-trivial subtlety about how one can prove security about Bloom-filter-based protocols.

One important role of a simulator is to extract a corrupt party's input. Consider the case of simulating the effect of a corrupt Bob. In the OT-hybrid model the simulator sees Bob's OT choice bits as well as the permutation π that he sends in 5. Hence, the simulator can easily extract Bob's "effective" Bloom filter. However, the simulator actually needs to extract the receiver's *input set* that corresponds to that Bloom filter, so that it can send the set itself to the ideal functionality.

In short, the simulator must *invert* the Bloom filter. While invertible Bloom filters do exist [GM11], they require storing a significant amount of data beyond that of a standard Bloom filter. Yet this PSI protocol only allows the simulator to extract the receiver's OT choice bits, which corresponds to a *plain* Bloom filter. Besides that, in our setting we must invert a Bloom filter that may not have been honestly generated.

We argue that the required kind of BF inversion *inherently* relies on non-standard-model assumptions. That is, (apparently) the only way the simulator can invert the receiver's Bloom filter is to treat the Bloom filter's hash functions as *random oracles.* In particular, the simulator needs the ability to *observe* the adversary's queries to the Bloom filter hash functions.[5] Let Q be the set of queries made by the adversary to any such hash function. This set has polynomial size, so the simulator can probe the extracted Bloom filter to test each $q \in Q$ for membership. The simulator can take the appropriate subset of Q as the adversary's extracted input set. More details are given in the security proof below.

To see why a random-oracle-type (ideal-model) assumption seems necessary, consider the problem of simulating a *semi-honest* Bob who has a single, randomly chosen item $y \in \{0,1\}^\kappa$ in its input set. Let S be the set of indices set to 1 in Bob's effective Bloom filter. Then the simulator must find a value v so that $h_i(v) \in S$ for all i, or else the simulation will clearly fail (since the environment might have arranged for Alice & Bob to always have the same item). If the BF hash functions are random oracles that the simulator *cannot* observe, then the simulator who makes q queries to these oracles has probability at most $q(|S|/N)^k$ of finding such a v, where N is the size of the BF. This probability is negligible. Hence, we conclude that the simulator must be allowed to observe the adversary's queries, or else we must assume some nonstandard property of the BF hash functions that facilitates such extraction.

Simulation/extraction of a corrupt Alice is also facilitated by observing her oracle queries. Recall that the *summary value* of x is (supposed to be) $H(x \| \bigoplus_{j \in h_*(x)} m_{\pi(j),1})$. Since H is a non-programmable random oracle, the simulator can obtain candidate x values from her calls to H.

More details about malicious Bloom filter extraction are given in the security proof in Section 5.3.

5.2 Cut-and-choose parameters

The protocol mentions various parameters:

N_{ot}: the number of OTs

N_{bf}: the number of Bloom filter bits

k: the number of Bloom filter hash functions

α: the fraction of 1s among Bob's choice bits

p_{chk}: the fraction of OTs to check

N_{maxones}: the maximum number of 1 choice bits allowed to pass the cut-and-choose.

As before, we let κ denote the computational security parameter and λ denote the statistical security parameter.

We require the parameters to be chosen subject to the following constraints:

- *The cut-and-choose restricts Bob to few 1s.* Let N_1 denote the number of OTs that remain after the cut and choose, in which Bob used choice bit 1. In the security proof we argue that the difficulty of finding an element stored in the Bloom filter *after the fact* is $(N_1/N)^k$ (i.e., one must find a value which all k random Bloom filter hash functions map to a 1 in the BF).

  Let $\mathcal{B}$ denote the "bad event" that no more than N_{maxones} of the checked OTs used choice bit one (so Bob can pass the cut-and-choose), and yet $(N_1/N_{\mathsf{bf}})^k \geq 2^{-\kappa}$. We require $\Pr[\mathcal{B}] \leq 2^{-\lambda}$.

  As mentioned above, the spirit of the protocol is to restrict a corrupt receiver from setting too many 1s in its (plain) Bloom filter. DCW suggest to restrict the receiver to 50% 1s, but do not explore how the fraction of 1s affects security (except to point out that 100% 1s is problematic). Our analysis pinpoints precisely how the fraction of 1s affects security.

- *The cut-and-choose leaves enough OTs unopened for the Bloom filter.* That is, when choosing from among N_{ot} items, each with independent p_{chk} probability, the probability that less than N_{bf} remain unchosen is at most $2^{-\lambda}$.

- *The honest Bob has enough one choice bits after the cut and choose.* When inserting n items into the bloom filter, at most nk bits will be set to one. We therefore require that no fewer than this remain after the cut and choose.

[5] The simulator does not, however, require the ability to *program* the random oracle.

Our main technique is to apply the Chernoff bound to the probability that Bob has too many 1s after the cut and choose. Let $m_h^1 = \alpha N_{\mathsf{ot}}$ (resp. $m_h^0 = (1-\alpha)N_{\mathsf{ot}}$) be the number of 1s (resp. 0s) Bob is supposed to select in the OT extension. Then in expectation, there should be $m_h^1 p_{\mathsf{chk}}$ ones in the cut and choose open set, where each OT message is opened with independent probability p_{chk}. Let ϕ denote the number of ones in the open set. Then applying the Chernoff bound we obtain,

$$\Pr[\phi \geq (1+\delta)m_h^1 p_{\mathsf{chk}}] \leq e^{-\frac{\delta^2}{2+\delta} m_h^1 p_{\mathsf{chk}}} \leq 2^{-\lambda}$$

where the last step bounds this probability to be negligible in the statistical security parameter λ. Solving for δ results in,

$$\delta \leq \frac{\lambda + \sqrt{\lambda^2 + 8\lambda m_h^1 p_{\mathsf{chk}}}}{2m_h^1 p_{\mathsf{chk}}}.$$

Therefore an honest Bob should have no more than $N_{\mathsf{maxones}} = (1+\delta)m_h^1 p_{\mathsf{chk}}$ 1s revealed in the cut and choose, except with negligible probability. To ensure there are at least nk ones[6] remaining to construct the bloom filter, set $m_h^1 = nk + N_{\mathsf{maxones}}$. Similarly, there must be at least N_{bf} unopened OTs which defines the total number of OTs to be $N_{\mathsf{ot}} = N_{\mathsf{bf}} + (1+\delta^*)N_{\mathsf{ot}} p_{\mathsf{chk}}$ where δ^* is analogous to δ except with respect to the total number of OTs opened in the cut and choose.

A malicious Bob can instead select $m_a^1 \geq m_h^1$ ones in the OT extension. In addition to Bob possibly setting more 1s in the BF, such a strategy will increase the probability of the cut and choose revealing more than N_{maxones} 1s. A Chernoff bound can then be applied to the probability of seeing a δ' factor fewer 1s than expected. Bounding this to be negligible in the statistical security parameter λ, we obtain,

$$\Pr[\phi \leq (1-\delta')p_{\mathsf{chk}} m_a^1] \leq e^{-\frac{\delta'^2}{2} p_{\mathsf{chk}} m_a^1} \leq 2^{-\lambda}.$$

Solving for δ' then yields $\delta' \leq \sqrt{\frac{2\lambda}{p_{\mathsf{chk}} m_a^1}}$. By setting N_{maxones} equal to $(1-\delta')p_{\mathsf{chk}} m_a^1$ we can solve for m_a^1 such that the intersection of these two distribution is negligible. Therefore the maximum number of 1s remaining is $N_1 = (1-p_{\mathsf{chk}})m_a^1 + \sqrt{2\lambda p_{\mathsf{chk}} m_a^1}$.

For a given p_{chk}, n, k, the above analysis allows us to bound the maximum advantage a malicious Bob can have. In particularly, a honest Bob will have at least nk 1s and enough 0s to construct the bloom filter while a malicious Bob can set no more than N_1/N_{bf} fraction of bits in the bloom filter to 1. Modeling the bloom filter hash function as random functions, the probability that all k index the boom filter one bits is $(N_1/N_{\mathsf{bf}})^k$. Setting this to be negligible in the computational security parameter κ we can solve for N_{bf} given N_1 and k. The overall cost is therefore $\frac{N_{\mathsf{bf}}}{(1-p_{\mathsf{chk}})}$. By iterating over values of k and p_{chk} we obtain set of parameters shown in Figure 5.

5.3 Security Proof

Theorem 1. *The protocol in Figure 4 is a UC-secure protocol for PSI in the random-OT-hybrid model, when H and the Bloom filter hash functions are non-programmable random oracles, and the other protocol parameters are chosen as described above.*

Proof. We first discuss the case of a corrupt receiver Bob, which is the more difficult case since we must not only extract Bob's input but simulate the output. The simulator behaves as follows:

> The simulator plays the role of an honest Alice and ideal functionalities in steps 1 through 5, but also extracts all of Bob's choice bits b for the OTs. Let N_1 be the number of OTs with choice bit 1 that remain after the cut and choose. The simulator artificially aborts if Bob succeeds at the cut and choose and yet $(N_1/N_{\mathsf{bf}})^k \geq 2^{-\kappa}$. From the choice of parameters, this event happens with probability only $2^{-\lambda}$.
>
> After receiving Bob's permutation π in step 5, the simulator computes Bob's effective Bloom filter $BF[i] = b_{\pi(i)}$. Let Q be the set of queries made by Bob to *any* of the Bloom filter hash

[6] nk ones is an upper bound on the number of ones required. A tighter analysis could be obtained if collisions were accounted for.

functions (random oracles). The simulator computes $\tilde{Y} = \{q \in Q \mid \forall i : BF[h_i(q)] = 1\}$ as Bob's effective input, and sends $\tilde{Y}$ to the ideal functionality. The simulator receives $Z = X \cap \tilde{Y}$ as output, as well as $|X|$. For $z \in Z$, the simulator generates $K_z = H(z \parallel \bigoplus_{j \in h_*(z)} m_{\pi(j),1})$. The simulator sends a random permutation of K_z along with $|X| - |Z|$ random strings to simulate Alice's message in step 6.

To show the soundness of this simulation, we proceed in the following sequence of hybrids:

1. The first hybrid is the real world interaction. Here, an honest Alice also queries the random oracles on her actual inputs $x \in X$. For simplicity later on, assume that Alice queries her random oracle as late as possible (in step 6 only).

2. In the next hybrid, we artifically abort in the event that $(N_1/N_{\mathsf{bf}})^k \geq 2^{-\kappa}$. As described above, our choice of parameters ensures that this abort happens with probability at most $2^{-\lambda}$, so the hybrids are indistinguishable.

   In this hybrid, we also observe Bob's OT choice bits. Then in step 5 of the protocol, we compute Q, BF, and $\tilde{Y}$ as in the simulator description above.

3. We next consider a sequence of hybrids, one for each item x of Alice such that $x \in X \setminus \tilde{Y}$. In each hybrid, we replace the summary value $K_x = H(x \parallel \bigoplus_{j \in h_*(x)} m_{\pi(j),1})$ with a uniformly random value.

   There are two cases for $x \in X \setminus \tilde{Y}$:

   - Bob queried some h_i on x before step 5: If this happened but x was not included in $\tilde{Y}$, then x is *not* represented in Bob's effective Bloom filter BF. There must be an i such that Bob did not learn $m_{\pi(h_i(x)),1}$.
   - Bob did *not* query any h_i on x: Then the value of $h_i(x)$ is random for all i. The probability that x is present in BF is the probability that $BF[h_i(x)] = 1$ for all i, which is $(N_1/N_{\mathsf{bf}})^k$ since Bob's effective Bloom filter has N_1 ones. Recall that the interaction is already conditioned on the event that $(N_1/N_{\mathsf{bf}})^k < 2^{-\kappa}$. Hence it is with overwhelming probability that Bob did not learn $m_{\pi(h_i(x)),1}$ for some i.

   In either case, there is an i such that Bob did not learn $m_{\pi(h_i(x)),1}$, so that value is random from Bob's view. Then the corresponding sum $\bigoplus_{j \in h_*(x)} m_{\pi(j),1}$ is uniform in Bob's view.[7] It is only with negligible probability that Bob makes the oracle query $K_x = H(x \parallel \bigoplus_{j \in h_*(x)} m_{\pi(j),1})$. Hence K_x is pseudorandom and the hybrids are indistinguishable.

In the final hybrid, the simulation does not need to know X, it only needs to know $X \cap \tilde{Y}$. In particular, the values $\{K_x \mid x \in X \setminus \tilde{Y}\}$ are now being simulated as random strings. The interaction therefore describes the behavior of our simulator interacting with corrupt Bob.

Now consider a corrupt Alice. The simulation is as follows:

The simulator plays the role of an honest Bob and ideal functionalities in steps 1 through 4. As such, the simulator knows Alice's OT outputs $m_{i,b}$ for all i, b, and can compute the correct r^* value in step 4. The simulator sends a completely random permutation π in step 5.

In step 6, the simulator obtains a set K as Alice's protocol message. Recall that each call made to random oracle H has the form $q \| s$. The simulator computes $Q = \{q \mid \exists s :$ Alice queried H on $q \| s\}$. The simulator computes $\tilde{X} = \{q \in Q \mid H(q \parallel \bigoplus_{j \in h_*(q)} m_{\pi(j),1}) \in K\}$ and sends $\tilde{X}$ to the ideal functionality as Alice's effective input. Recall Alice receives no output.

[7] This is part of the proof that breaks down if we compute a summary value using $\bigoplus_i m_{\pi(h_i(x)),1}$ instead of $\bigoplus_{j \in h_*(x)} m_{\pi(j),1}$. In the first expression, it may be that $h_{i'}(x) = h_i(x)$ for some $i' \neq i$ so that the randomizing term $m_{\pi(h_i(x)),1}$ cancels out in the sum.

It is straight-forward to see that Bob's protocol messages in steps 4 & 5 are distributed independently of his input.

Recall that Bob outputs $\{y \in Y \mid H(y \parallel \bigoplus_{j \in h_*(y)} m^*_{\pi(j)}) \in K\}$ in the last step of the protocol. In the ideal world (interacting with our simulator), Bob's output from the functionality is $\tilde{X} \cap Y = \{y \in Y \mid y \in \tilde{X}\}$. We will show that the two conditions are the same except with negligible probability. This will complete the proof.

We consider two cases:

- If $y \in \tilde{X}$, then $H(y \parallel \bigoplus_{j \in h_*(y)} m^*_{\pi(j)}) = H(y \parallel \bigoplus_{j \in h_*(y)} m_{\pi(j),1}) \in K$ by definition.
- If $y \notin \tilde{X}$, then Alice never queried the oracle $H(y \| \cdot)$ before fixing K, hence $H(y \parallel \bigoplus_{j \in h_*(y)} m^*_{\pi(j)})$ is a fresh oracle query, distributed independently of K. The output of this query appears in K with probability $|K|/2^\kappa$.

Taking a union bound over $y \in Y$, we have that, except with probability $|K||Y|/2^\kappa$,

$$H(y \parallel \textstyle\bigoplus_{j \in h_*(y)} m^*_{\pi(j)}) \in K \iff y \in \tilde{X}$$

Hence Bob's ideal and real outputs coincide. $\square$

Size of the adversary's input set. When Alice is corrupt, the simulator extracts a set $\tilde{X}$. Unless the adversary has found a collision under random oracle H (which is negligibly likely), we have that $|\tilde{X}| \leq |K|$. Thus the protocol enforces a straightforward upper bound on the size of a corrupt Alice's input.

The same is not true for a corrupt Bob. The protocol enforces an upper bound only on the size on Bob's *effective Bloom filter* and a bound on the number of 1s in that BF. We now translate these bounds to derive a bound on the size of the set extracted by the simulator. Note that the ideal functionality for PSI (Figure 2) explicitly allows corrupt parties to provide larger input sets than honest parties.

First, observe that only queries made by the adversary before step 5 of the protocol are relevant. Queries made by the adversary *after* do not affect the simulator's extraction. As in the proof, let Q be the set of queries made by Bob before step 5. Bob is able to construct a BF with at most N_1 ones, and causing the simulator to extract items $\tilde{Y} \subseteq Q$, only if:

$$\left| \bigcup_{y \in \tilde{Y}; i \in [k]} h_i(y) \right| \leq N_1.$$

Then by a union bound over all Bloom filters with N_1 bits set to 1, and all $\tilde{Y} \subseteq Q$ of size $|\tilde{Y}| = n'$, we have:

$$\Pr\left[\begin{array}{c}\text{simulator extracts}\\ \text{some set of size } n'\end{array}\right] \leq \binom{|Q|}{n'}\binom{N_{\mathsf{bf}}}{N_1}\left(\frac{N_1}{N_{\mathsf{bf}}}\right)^{kn'}.$$

The security proof already conditions on the event that $(N_1/N_{\mathsf{bf}})^k \leq 2^{-\kappa}$, so we get:

$$\begin{aligned}\Pr\left[\begin{array}{c}\text{simulator extracts}\\ \text{some set of size } n'\end{array}\right] &\leq \binom{|Q|}{n'}\binom{N_{\mathsf{bf}}}{N_1}2^{-\kappa n'}\\ &\leq \left(|Q|^{n'}\right)\left(2^{N_{\mathsf{bf}}}\right)2^{-\kappa n'}\end{aligned}$$

To make the probability less than $2^{-\kappa}$ it therefore suffices to have $n' = (\kappa + N_{\mathsf{bf}})/(\kappa - \log|Q|)$.

In our instantiations, we always have $N_{\mathsf{bf}} \leq 3\kappa n$, where n denotes the *intended* size of the parties' sets. Even in the pessimistic case that the adversary makes $|Q| = 2^{\kappa/2}$ queries to the Bloom filter hash functions, we have $n' \approx 6n$. Hence, the adversary is highly unlikely to produce a Bloom filter containing 6 times the intended number of items. We emphasize that this is a very loose bound, but show it just to demonstrate that the simulator indeed extracts from the adversary a modestly sized effective input set.

5.4 Programmable Random Oracles in the UC Model

Our protocol makes significant use of a non-programmable random oracle. In the standard UC framework [Can01], the random oracle must be treated as *local* to each execution for technical reasons. The UC framework does not deal with global objects like a single random oracle that is used by many protocols/instances. Hence, as currently written, our proof implies security when instantiated with a highly local random oracle.

Canetti, Jain, & Scafuro [CJS14] proposed a way to model global random oracles in the UC framework (we refer to their model as UC-gRO). One of the main challenges is that (in the plain UC model) the simulator can observe the adversary's oracle queries, but an adversary can ask the environment to query the oracle on its behalf, hidden from the simulator. In the UC model, every functionality and party in the UC model is associated with a *session id* (sid) for the protocol instance in which it participates. The idea behind UC-gRO is as follows:

- There is a functionality gRO that implements an ideal random oracle. Furthermore, this functionality is **global** in the sense that all parties and all functionalities can query it.
- Every oracle query in the system must be prefixed with some sid.
- There is no enforcement that oracle queries are made with the "correct" sid. Rather, if a party queries gRO with a sid that does not match its own, that query is marked as **illegitimate** by gRO.
- A functionality can ask gRO for all of the illegitimate queries made using that functionality's sid.

Our protocol and proof can be modified in the following ways to provide security in the UC-gRO model:

1. In the protocol, all queries to relevant random oracles (Bloom filter functions h_i and outer hash function H) are prefixed with the sid of this instance.
2. The ideal PSI functionality is augmented in a standard way of UC-gRO: When the adversary/simulator gives the functionality a special command `illegitimate`, the functionality requests the list of illegitimate queries from gRO and forwards them to the adversary/simulator.
3. In the proof, whenever the simulator is described as obtaining a list of the adversary's oracle queries, this is done by observing the adversary's queries and also obtaining the illegitimate queries via the new mechanism.

With these modifications, our proof demonstrates security in the UC-gRO model.

6 Performance Evaluation

We implemented our protocol in addition to the protocol of DCW [DCW13] outlined Section 3 and report on their performance in this section.

6.1 Implementation & Test Platform

In the offline phase, our protocol consists of performing 128 base OTs using the protocol of [NP01]. We extend these base OTs to N_{ot} OTs using an optimized implementation of the Keller et al. [KOS15] OT extension protocol. In the multi-threaded case, the OT extension and Base OTs are performed in parallel. Subsequently, the cut and choose seed is published which determines the set of OT messages to be opened. Then one or more threads reports the choice bits used for the corresponding OT and the XOR sum of the messages. The sender validates the reported value and proceeds to the online phase.

The online phase begins with both parties inserting items into a plaintext bloom filter using one or more threads. As described in section 5.1, the BF hash functions should be modeled as (non-programmable) random oracles. We use SHA1 as a random oracle but then expand it to a suitable length via a fast PRG (AES in counter mode) to obtain:[8]

$$h_1(x)\|h_2(x)\|\cdots\|h_k(x) = \mathsf{PRG}(\mathsf{SHA1}(x)).$$

Hence we use just one (slow) call to SHA to compute all BF hash functions for a single element, which significantly reduces the time for generating Bloom filters. Upon the computing the plaintext bloom filter, the receiver selects a random permutation mapping the random OT choice bits to the desired bloom filter. The permutation is published and the sender responds with the random garbled bloom filter masks which correspond to their inputs. Finally, the receiver performs a plaintext intersection of the masks and outputs the corresponding values.

We evaluated the prototype on a single server with simulated network latency. The server has 2 36-cores Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz and 256GB of RAM (e.i. 36 cores & 128 GB per party). We executed our prototype in two network settings: a LAN configuration with both parties in the same network with 0.2 ms round-trip latency, 400 Mbps; and a WAN configuration with a simulated 95 ms round-trip latency, 40 Mbps. All experiments we performed with a computational security parameter of $\kappa = 128$ and statistical security parameter $\lambda = 40$. The times reported are an average over 10 trials. The variance of the trials was between $0.1\% - 5.0\%$ in the LAN setting and $0.5\% - 10\%$ in the WAN setting with a trend of smaller variance as n becomes larger. The CPUs used in the trials had AES-NI instruction set for fast AES computations.

6.2 Parameters

We demonstrate the scalability of our implementation by evaluating a range of set sizes $n \in \{2^8, 2^{12}, 2^{16}, 2^{20}\}$ for strings of length $\sigma = 128$. In all of our tests, we use system parameters specified in Figure 5. The parameters are computed using the analysis specified in Section 5.2. Most importantly they satisfy that except with probability negligible in the computation security parameter κ, a receiver after step 5 of Figure 4 will not find an x not previously queried which is contained in the garbled bloom filter.

The parameters are additionally optimized to reduce the overall cost of the protocol. In particular, the total number of OTs computed $N_{\mathsf{ot}} = N_{\mathsf{bf}}/(1 - p_{\mathsf{chk}})$ is minimized. This value is derived by iterating over all the region of $80 \le k \le 100$ hash functions and cut-and-choose probabilities $0.001 \le p_{\mathsf{chk}} \le 0.1$. For a given value of n, k, p_{chk}, the maximum number of ones N_1 which a possibly malicious receiver can have after the cut and choose is defined as shown in Section 5.2. This in turn determines the minimum value of N_{bf} such that $(N_{\mathsf{bf}}/N_1)^{-k} \le 2^{-\kappa}$ and therefore the overall cost N_{ot}. We note that for κ other than 128, a different range for the number of hash function may be optimal.

n	p_{chk}	k	N_{ot}	N_{bf}	α	N_{maxones}
2^8	0.099	94	99,372	88,627	0.274	3,182
2^{12}	0.053	94	1,187,141	1,121,959	0.344	22,958
2^{16}	0.024	91	16,992,857	16,579,297	0.360	150,181
2^{20}	0.010	90	260,252,093	257,635,123	0.366	962,092

Figure 5: Optimal Bloom filter cut and choose parameters for set size n to achieve statistical security $\lambda = 40$ and computational security $\kappa = 128$. N_{ot} denotes the total number of OTs used. N_{bf} denotes the bit count of the bloom filer. α is the faction of ones which should be generated. N_{maxones} is the maximum number of ones in the cut and choose to pass.

6.3 Comparison to Other Protocols

For comparison, we implemented two other protocol paradigms, which we describe here:

[8] Note that if we model SHA1 as having its queries observable to the simulator, then this property is inherited also when expanding the SHA1 output with a PRG.

Setting	Protocol	Threads	Set size n 2^8	2^{12}	2^{16}	2^{20}
LAN	*DCW (Fig. 3)	1	$1,558+1,397$	$30,669+27,791$	$602,823+531,596$	-
	*DCW + RGBF	1	$1,546+1,383$	$30,840+27,605$	$602,645+542,632$	-
	*DH-based ECC	1	$0+102$	$0+1,638$	$0+26,214$	$0+419,430$
	Ours (Fig 4)	1	$\mathbf{342+5}$	$\mathbf{1,266+75}$	$\mathbf{14,250+720}$	$\mathbf{183,622+14,312}$
		4	$321+5$	$613+64$	$4,588+670$	$56,677+9,930$
		32	$301+5$	$440+47$	$3,781+557$	$43,548+7,583$
		64	$\mathbf{301+5}$	$\mathbf{360+46}$	$\mathbf{2,460+472}$	$\mathbf{32,530+6,389}$
WAN	*DCW (Fig. 3)	1	$2,430+1,809$	$32,581+28,757$	$652,491+532,567$	-
	*DCW + RGBF	1	$2,381+1,609$	$32,011+28,601$	$659,750+530,168$	-
	Ours (Fig 4)	1	$1,067+189$	$2,020+501$	$15,021+5,320$	$222,950+79,210$
		64	$998+187$	$1,122+459$	$2,934+4,518$	$34,712+68,159$

Figure 6: Offline+Online running time in ms for two sets size n over elements of 128 bits. The LAN (resp. WAN) setting has 0.2ms (resp. 95ms) latency. As noted in Section 6.3, when the protocol is marked with an asterisk, we report an optimistic underestimate of the running time. Missing times (-) took > 5 hours.

	set size n 2^8	2^{12}	2^{16}	2^{20}	asymptotic offline	online
DCW (Fig. 3)	3.2	50.7	810	-	$2n\kappa^2$	$4nk^2$
DCW + RGBF	2.4	33.9	541	-	$2n\kappa^2$	$nk+2n\kappa^2$
DH-based ECC	0.01	0.43	6.9	106	0	$3n\phi$
Ours (Fig 4)	2.3	29.5	407	6,620	$N_{\mathsf{ot}}\kappa \approx 2n\kappa^2$	$N_{\mathsf{bf}} \log N_{\mathsf{ot}} + n\kappa$

Figure 7: Communication overhead in MB for PSI protocols with sets of size n. $\phi = 283$ is the size of the elliptic curve elements. Missing entries had prohibitively long running times and are estimated to be greater than $8,500$MB.

DCW protocol Our first point of comparison is to the protocol of Dong, Chen, & Wen [DCW13], on which ours is based. The protocol is described in Section 3. While their protocol has issues with its security, our goal here is to illustrate that our protocol also has significantly better performance.

In [DCW13], the authors implement only their semi-honest protocol variant, not the malicious one. An aspect of the malicious DCW protocol that is easy to overlook is its reliance on an $N/2$-out-of-N secret sharing scheme. When implementing the protocol, it becomes immediately clear that such a secret-sharing scheme is a major computational bottleneck.

Recall that the sender generates shares from such a secret sharing scheme, and the receiver reconstructs such shares. In this protocol, the required N is the number of bits in the Bloom filter. As a concrete example, for PSI of sets of size 2^{20}, the Bloom filter in the DCW protocol has 2^{28} bits. Using Shamir secret sharing, the sender must evaluate a random polynomial of degree $2^{27}-1$ on 2^{28} points. The sender must interpolate such a polynomial on 2^{27} points to recover the secret. Note that the polynomial will be over $GF(2^{128})$, since the protocol secret-shares an (AES) encryption key.

We chose not to develop a full implementation of the malicious DCW protocol. Rather, we fully implemented the [garbled] Bloom filter encoding steps and the OTs. We then **simulated** the secret-sharing and reconstruction steps in the following way. We calculated the number of field multiplications that would be required to evaluate a polynomial of the suitable degree by the Fast Fourier Transform (FFT) method, and simply had each party perform the appropriate number of field multiplications in $GF(2^{128})$. The field was instantiated using the NTL library with all available optimization enabled. Our simulation significantly underestimates the cost of secret sharing in the DCW protocol, since: (1) it doesn't account for the cost associated with virtual memory accesses when computing on such a large polynomial; and (2) evaluating/interpolating the polynomial via FFT reflects a *best-case scenario*, when the points of evaluation are roots of unity. In the protocol, the receiver Bob in particular does not have full control over which points of the polynomial he will learn.

Despite this optimistic simulation of the secret-sharing step, its cost is substantial, accounting for 97% of the execution time. In particular, when comparing our protocol to the DCW protocol, the main difference in the online phase is the secret sharing reconstruction which accounts for a 113× increases in the online running time for $n = 2^{16}$.

We simulated two variants of the DCW malicious-secure protocol. One variant reflects the DCW protocol as written, using OTs of chosen messages. The other variant includes the "random GBF" optimization inspired by [PSZ14] and described in Section 4. In this variant, one of the two OT messages is set randomly by the protocol itself, and not chosen by the sender. This reduces the online communication cost of the OTs by roughly half. However, it surprisingly has a slight negative effect on total time. The reason is that Alice has more than enough time to construct and send a plain GBF while Bob performs the more time intensive secret-share reconstruction step. For $n = 2^{16}$, the garbled bloom filter takes less than 5% of the secret share reconstruction time to be sent. When using a randomized GBF, Alice sends summary values to Bob, which he must compare to his own summary values. Note that there is a summary value for each item in a party's set (e.g., 2^{20}), so these comparisons involve lookups in some non-trivial data structure. This extra computational effort is part of the the critical path since the Bob has to do it.

DH-based PSI protocols Another paradigm for PSI uses public-key techniques and is based on Diffie-Hellman-type assumptions in cyclic groups. The most relevant protocol in this paradigm that achieves malicious security is that of De Cristofaro, Kim, and Tsudik [DKT10]. While protocols in this paradigm have extremely low communication complexity, they involve a large number of computationally expensive public-key operations (exponentiations).

Rather than fully implement a malicious-secure, DH-based PSI protocol, we again chose to *simulate* its cost. In fact, we simulate the cost of the most basic *semi-honest* PSI protocol of Huberman, Franklin, and Hogg (HFH) [HFH99]. In this protocol, each party performs $2n$ exponentiations in a cyclic group, where n is the size of each party's set. Any malicious-secure protocol in this Diffie-Hellman paradigm must be at least this expensive, so we use the HFH protocol as a generous baseline.

In Figure 6 we include the computational cost of $2n$ exponentiations on our hardware. The Miracl library elliptic curve implementation was used with the NIST Koblitz Curves K283 achieving 128 bit computational security. The benchmarking machine performed approximately 5 exponentiations per ms. These numbers do not include any network latency or other memory costs associated with a concrete protocol. Any malicious-secure PSI protocol will almost certainly involve more computation than $2n$ exponentiations. Still, the cost of this generous baseline is much more than the cost of our malicious-secure protocol.

6.4 Results

The running time of our implementation is shown in Figure 6. We make the distinction of reporting the running times for both the online and offline phases of the protocols. The offline phase contains all operation which are independent of the input sets. For the bloom filter based protocols the offline phase consists of performing the OT extension and the cut and choose. As expected, our optimized protocol achieves the smallest online running times. When executing the program with $n = 2^8$ on a single thread in the LAN setting, the online running is only $5ms$. For larger set sizes, our protocol achieves the smallest online and overall running times. For $n = 2^{12}$, the overall running time is $1,341ms$ and only $75ms$ for the online phase. The next fastest protocol is the estimated time for the DH-based PSI with $1,638ms$ in the online phase. For the largest set size performed of $n = 2^{20}$, our protocol achieves an online phase of $14,312ms$ and an overall time of $197,934ms$. The DH-based protocol achieves an online running time more than 2× times longer.

When evaluating our protocol in the WAN setting with $95ms$ round trip latency our protocol again achieves the smallest running times. For the small set size of $n = 2^8$, the protocol takes an overall running time of $1,256ms$ with the online phase taking $189ms$. The next faster protocol tested was the DCW protocol with the randomized garbled bloom filter optimization suggested by [PSZ14]. This optimized DCW protocol takes an overall time of $4,239ms$ and $1,809ms$ in the online phase. When scaling to larger set sizes, the effects of the network latency diminish. For instance, with set size $n = 2^{20}$ our offline performance decreases only slightly and online performance requires roughly 5× times longer when executing with a single thread.

In the offline phase, the most timing consuming operation is the OT extension. For instance, with $n = 2^{20}$ the OT extension takes 180 seconds and the cut and choose takes only 3 seconds. For the smaller set size of $n = 2^{12}$, the OT extension required $582ms$ and the cut and choose completed in $546ms$. The relative increase in the cut and choose running time is primarily due to the need to open a larger portion of the OTs when n is smaller.

The online phase consists of the receiver first computing their bloom filter. For set size $n = 2^{20}$, computing the bloom filter takes 6.4 seconds. The permutation mapping the receiver's OTs to the bloom filter then computed in less than a second and sent. Upon receiving the permutation, the sender computes their PSI summary values and sends them to the receiver. This processes when $n = 2^{20}$ takes roughly 6 seconds. The receiver then outputs the intersection in less than a second.

In addition to faster serial performance, our protocol also benefits from easily being parallelized, unlike much of the DCW online phase. The first three rows of Figure 6 contain the running times of our protocol when parallelized using p threads per party in the LAN setting. For set size $n = 2^{16}$ and $p = 4$, we obtain a speedup of 3.1× in the offline phase. For the largest size of $n = 2^{20}$, the speedup of $p = 4$ threads is 1.4× for the online phase and 3.2× for the offline phase. When using $p = 64$ threads, the online speedup is 2.2× and the offline speedup is 5.6×. The online phase benefits less from $p = 64$ due to it being primarily IO bound.

In *Figure* 7 we report the empirical and asymptotic communication costs of the protocols. Out of the bloom filter based protocols, ours consume significantly less bandwidths. For $n = 2^8$, only 2.3MB communication was consumed with the largest portion being performed in the offline phase. Then computing the intersection for $n = 2^{16}$, our protocol consumes 407MB of communication, approximately 6KB per item. The largest amount of communication occurs during the OT extension and involves the sending of a $n\kappa^2$ bit matrix. The cut and choose contributes minimally to the communication and consists of np_{chk} choice bits and the xor of the corresponding OT messages. In the online phase, the sending of the permutation consisting of $N_{\mathsf{bf}} \log_2(N_{\mathsf{ot}})$ bits then dominates the communication, where $N_{\mathsf{bf}} \approx N_{\mathsf{ot}} \approx n\kappa$.

References

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In Sadeghi et al. [SGY13], pages 535–548.

[ALSZ15] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 673–701. Springer, Heidelberg, April 2015.

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *28th ACM STOC*, pages 479–488. ACM Press, May 1996.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14*, pages 597–608. ACM Press, November 2014.

[DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In Sadeghi et al. [SGY13], pages 789–800.

[DKT10] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 213–231. Springer, Heidelberg, December 2010.

[DMRY09] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS 09*, volume 5536 of *LNCS*, pages 125–142. Springer, Heidelberg, June 2009.

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 1–19. Springer, Heidelberg, May 2004.

[GM11] Michael T. Goodrich and Michael Mitzenmacher. Invertible bloom lookup tables. In *49th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2011, Allerton Park & Retreat Center, Monticello, IL, USA, 28-30 September, 2011*, pages 792–799. IEEE, 2011.

[HEK12] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS 2012*. The Internet Society, February 2012.

[HFH99] Bernardo A. Huberman, Matthew K. Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *EC*, pages 78–86, 1999.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, August 2003.

[KMRS14] Seny Kamara, Payman Mohassel, Mariana Raykova, and Seyed Saeed Sadeghian. Scaling private set intersection to billion-element sets. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 195–215. Springer, Heidelberg, March 2014.

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 724–741. Springer, Heidelberg, August 2015.

[Lam16] Mikkel Lambæk. Breaking and fixing private set intersection protocols. Master's thesis, Aarhus University, 2016. `https://eprint.iacr.org/2016/665`.

[Lin13] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 1–17. Springer, Heidelberg, August 2013.

[Mar14] Moxie Marlinspike. The difficulty of private contact discovery, 2014. Blog post, whispersystems.org/blog/contact-discovery.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In S. Rao Kosaraju, editor, *12th SODA*, pages 448–457. ACM-SIAM, January 2001.

[PSSZ15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In Jaeyeon Jung and Thorsten Holz, editors, *24th USENIX Security Symposium, USENIX Security 15*, pages 515–530. USENIX Association, 2015.

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In Kevin Fu and Jaeyeon Jung, editors, *23rd USENIX Security Symposium, USENIX Security 14*, pages 797–812. USENIX Association, 2014.

[Riv99] Ronald L. Rivest. Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer, 1999. Unpublished manuscript. people.csail.mit.edu/rivest/Rivest-commitment.pdf.

[SGY13] Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors. *ACM CCS 13*. ACM Press, November 2013.