Ágnes Kiss*, Jian Liu, Thomas Schneider, N. Asokan, and Benny Pinkas

Private Set Intersection for Unequal Set Sizes with Mobile Applications

Abstract: Private set intersection (PSI) is a cryptographic technique that is applicable to many privacy-sensitive scenarios. For decades, researchers have been focusing on improving its efficiency in both communication and computation. However, most of the existing solutions are inefficient for an unequal number of inputs, which is common in conventional client-server settings. In this paper, we analyze and optimize the efficiency of existing PSI protocols to support precomputation so that they can efficiently deal with such input sets. We transform four existing PSI protocols into the precomputation form such that in the setup phase the communication is linear only in the size of the larger input set, while in the online phase the communication is linear in the size of the smaller input set. We implement all four protocols and run experiments between two PCs and between a PC and a smartphone and give a systematic comparison of their performance. Our experiments show that a protocol based on securely evaluating a garbled AES circuit achieves the fastest setup time by several orders of magnitudes, and the fastest online time in the PC setting where AES-NI acceleration is available. In the mobile setting, the fastest online time is achieved by a protocol based on the Diffie-Hellman assumption.

Keywords: Private set intersection, Bloom filter, oblivious pseudorandom function

DOI 10.1515/popets-2017-0044
Received 2017-02-28; revised 2017-06-01; accepted 2017-06-02.

***Corresponding Author: Ágnes Kiss:** TU Darmstadt, E-mail: agnes.kiss@crisp-da.de
Jian Liu: Aalto University, E-mail: jian.liu@aalto.fi
Thomas Schneider: TU Darmstadt, E-mail: thomas.schneider@crisp-da.de
N. Asokan: Aalto University and University of Helsinki, E-mail: asokan@acm.org
Benny Pinkas: Bar Ilan University, E-mail: benny@pinkas.net

1 Introduction

Private set intersection (PSI) enables two parties to compute the intersection of their inputs in a privacy-preserving way, such that only the common inputs are revealed. PSI has been used in many privacy-sensitive scenarios such as common friends discovery [47], fully-sequenced genome test [5], online matching [24], collaborative botnet detection [46], location sharing [49] as well as measuring the rate of converting ad viewers to customers [53].

In the above mentioned scenarios, both parties have an equal number of inputs, i.e., the input sets have similar sizes. However, an unequal number of inputs commonly appears in client-server settings, where the server's data set is significantly larger than that of the client, such as applications in the mobile setting. Moreover, in many scenarios, the server's large database may undergo frequent updates, which poses another challenge for the design of PSI protocols when unequal set sizes are considered. In these scenarios, the client also has limited computational power and storage capacity compared to the server.

In the following, we detail example application scenarios where the client's input set is significantly smaller than the server's input set, which may be frequently updated. In our scenarios, since the database can be a trade secret of a provider and thus a business advantage, it should be kept private (as in [58, 60]).

Mobile malware detection service. An antivirus company which holds a large malware database wants to provide a malware detection service in a privacy-preserving way, i.e., an end-user can use this service to check a small set of applications such that the company learns nothing about these applications and the user learns nothing about the database except for the detected malware. Here, the number of malware signatures in the database is significantly larger than the number of applications on the user's device. Naturally, malware signatures need to be frequently inserted to or removed from the server's database. A similar application was presented in [60] where a solution using trusted hardware was proposed.

Mobile messaging service. A messaging service for mobile devices has a large set of users, stored in a server-side database. Once it is installed on a mobile device, the user wants to check if anyone on its contact list is using that messaging service, i.e., if any of the user's contacts are in the database. The number of contacts of the user is significantly smaller than that stored in the database. The messaging service should allow new users to register and old users to unregister.

Discovery of leaked passwords. A database storing breached or stolen passwords could provide a service that allows a user to detect if its passwords are among the stolen ones stored in the database or not. In this case, the number of passwords used by the client is significantly smaller than the number of entries in the database, which might be updated with new password leaks continuously. A scenario where the account name is stored instead of the stolen passwords can also be considered. Here, even though the account name of the user might be public, revealing the different account names of the same user might violate its privacy.

Search for chemical compound databases. In [58], the authors consider a scenario where similar compounds are to be recovered by searching a database for an existing chemical compound. However, in certain scenarios we can assume that only exact matches are interesting. Then, private set intersection for unequal number of inputs and efficient updates can provide a more efficient solution than the protocol in [58].

A PSI scheme with sublinear complexity in both computation and communication seems to be impossible since the server has to touch the whole database. Otherwise, the server would learn that the elements left untouched are not in the client's input set which would compromise the client's privacy. There are promising research directions such as random-access machine (RAM) model secure computation (e.g., [28]), fully homomorphic encryption (e.g., [25]), or protocols based on (symmetric/single-server) private information retrieval (PIR) (e.g., [41]), that may provide solutions with sub-linear complexity if adapted to the offline/online setting. However, currently all these solutions have substantially higher constant factors than the protocols studied in this work.[1]

Three phases of the protocols. In this paper, we propose a practical solution to the problem described above by exploiting the *offline/online* characteristics of the scenarios of interest, similarly to the strategy of [13] for generic secure two-party computation. Throughout this paper, we refer to the following three phases for our different protocols: the *base phase* where data-independent precomputation takes place; the *setup phase* with communication linear in the server's larger set, and the *online phase* with communication linear in the client's smaller set. Moreover, our main aim is to shift most of the communication and computation into the base and setup phases and have a very efficient online phase for unequal number of inputs with efficient secure updates in the database, since the other two phases need to be run once and the online phase is required for each query.

By having the server transfer $\mathcal{O}(N_S)$ data once in a setup phase, where N_S is the size of the the server's database, we are able to shift the bulk of the communication offline. After this, communication and computation linear only in the size of the client's input set N_C is required in the online phase, which is assumed to be significantly smaller than N_S. For instance, in our first motivating scenario, after installing the service provided by the malware database, the user may run the base and setup phases overnight. Then, by running the online phase, the client can get access to the list of installed malwares on its device within a short period of time. Thereafter, efficient updates (i.e., insertion or deletion) on the database can be performed without recomputing the base and setup phases. An update in the client's input set requires running only the efficient online phase.

1.1 Our Contributions

In this paper we investigate the efficiency of PSI protocols in the precomputation setting, especially when one party has significantly more inputs than the other. More detailed, our contributions are as follows:

1 All known single-server PIR constructions require the server to compute for each query one public-key operation per item in its database. Therefore, even though communication might be small, the computation overhead of the server is $O(N_S)$ expensive operations per query and downloading the database is in many cases more efficient than PIR [59].

Improving existing PSI protocols (§2). We investigate four existing PSI protocols with linear communication complexity: RSA-based PSI (RSA-PSI) of [11], Diffie-Hellman-based PSI (DH-PSI) of [35], Naor-Reingold OPRF-based PSI (NR-PSI) of [31, 48], and PSI using AES evaluated with a garbled circuit (GC-PSI) of [54]. We show that these protocols can be used in the setting of PSI with an unequal number of inputs such that the complexity in the online phase depends only on the size of the client's small input set. We describe how the larger input set can be updated efficiently without running the setup phase again. Moreover, we extend these protocols by using Bloom filters to reduce the communication and storage overhead.

Experimental comparison (§3). We implemented all four PSI protocols and systematically compare their performance. We built a prototype for the client's application both on PC and on an Android platform. To the best of our knowledge, this is the first comparison of PSI protocols with linear complexity on a smartphone. Our experiments show that the protocol based on the secure evaluation of a garbled AES circuit achieves the best overall performance but requires the most online communication and client storage capacity. Its setup phase is orders of magnitude more efficient than that of any other protocol, since it employs only very efficient AES evaluations on the server's large database. Its online phase is also the most efficient in our PC implementation using hardware accelerated AES, while in the smartphone setting the protocol based on the Diffie-Hellman assumption is more efficient. Our results on PC indicate that advancements on hardware-accelerated encryption on smartphones could greatly improve the performance of PSI with unequal set sizes.

Further extensions (§4). We show that some of the protocols can serve multiple clients over a broadcast communication channel or a content distribution network and can easily be secured against malicious clients.

1.2 Related Work on PSI

Among the first protocols for PSI was the PSI protocol of [23] which is based on Oblivious Polynomial evaluation (OPE). However, this protocol requires $(N_S)^2$ computationally heavy public-key operations in the online phase. In this work, we are interested in protocols with linear computation and communication complexity.

The first PSI protocol with linear computation and communication complexity was proposed in [44], and is based on the Diffie-Hellman protocol (DH).

PSI using *oblivious pseudorandom function* (OPRF) evaluation was proposed in [22, 31], where the Naor-Reingold (NR) pseudorandom function [48] was used. In this protocol, $\mathcal{S}$ randomly chooses a symmetric key k and sends $\mathrm{PRF}_k(\mathbf{x}_i)$ for all its elements $\mathbf{x}_i \in \mathcal{X}$ to $\mathcal{C}$. Then, they invoke an OPRF protocol, where $\mathcal{C}$ inputs its elements $\mathbf{y}_i \in \mathcal{Y}$, $\mathcal{S}$ inputs k and $\mathcal{C}$ obliviously obtains $\mathrm{PRF}_k(\mathbf{y}_i)$ as output. Using these values, $\mathcal{C}$ can compute the intersection. A variant of this protocol where the PRF is instantiated with AES has been proposed in [54]. The OPRF can also be instantiated using RSA blind signatures [11].

Today's most recent and most efficient PSI protocols are based on efficient OT extension and use either garbled Bloom filters [18] or hashing to bins [39, 53, 55, 56]. The basic idea of all OT-based PSI protocols is having $\mathcal{S}$ and $\mathcal{C}$ run a random OT for each bit of $\mathcal{C}$'s input y_i, such that $\mathcal{S}$ gets two random values and $\mathcal{C}$ gets one of them corresponding to its input bits. Then, both of them XOR the random values for each of their input elements. $\mathcal{S}$ sends the results to $\mathcal{C}$, who locally checks the existence of its inputs. The data sent by $\mathcal{S}$ is linear in the size of its input set, and it must be sent for each query since the randomness can be used only once. Therefore, such protocols are not suitable for the online/offline setting.

Existing PSI protocols are compared in [56], where experiments are performed for both equal and unequal number of inputs. We reviewed the different PSI protocols surveyed in [55, 56] for their adaptability in our setting, i.e., if they can be transformed to have an online phase dependent only on one of the parties' inputs. Most of the existing protocols require linear work in the size of both sets for each query and therefore are not adaptable for our setting, as depicted in Tab. 1.

Implementations of PSI protocols on smartphones such as [4, 10, 33] can be found in the literature, but they either do not achieve linear complexity or do not consider the offline/online setting, and hence are not suited for our scenario.

Our work is similar to [50], where protocols were instantiated using RSA blind signatures and the Naor-Reingold OPRF. Our RSA-PSI is an improvement over their RSA-based protocol by shifting all possible computation offline in order to achieve a more efficient online phase. Our NR-PSI is different from their Naor-Reingold OPRF-based protocol since they need to run multiple OPRF instances to calculate the Bloom filter

Type	Protocol	Reference	Adaptable
Public-key	OPE	[23]	×
	RSA	[11]	✓ (§2.1)
	DH	[44]	✓ (§2.2)
	NR-PRF	[31]	✓ (§2.3)
Circuit	Sort-Compare-Shuffle	[34]	×
	Circuit-Phasing	[53]	×
	AES-OPRF	[54]	✓ (§2.4)
OT	Garbled BF	[18]	×
	Random Garbled BF	[55]	×
	OT + Hashing	[55]	×

Table 1. PSI protocols surveyed in [55, 56] and their adaptability to our setting. We mark with ✓ if a protocol can be modified such that its online complexity only depends on the size of one of the input sets and with × otherwise.

$\mathcal{S}$	Server party, with large input set and computational power
$\mathcal{C}$	Client party, with significantly smaller data set and computational power
N_S	Number of server inputs
N_C	Number of client inputs
$N_C^{\max}$	Maximal number of client inputs, $N_C \leq N_C^{\max}$
N_u	Number of server inputs for update
$\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_{N_S}\}$	Server inputs
$\mathcal{Y} = \{\mathbf{y}_1, ..., \mathbf{y}_{N_C}\}$	Client inputs
$\mathbf{x}_i$ or $\mathbf{y}_i$	i^{th} input
$\mathbf{x}_i[j]$ or $\mathbf{y}_i[j]$	j^{th} bit of i^{th} input
n	Number of bits of inputs $\mathbf{x}_i$ or $\mathbf{y}_i$
m	Bitlength of the ciphertext in protocols
BF	Bloom filter of length $1.44\epsilon N_S$ bits
$BF.\mathit{Insert}(e)$	Insert element e in BF.
$BF.\mathit{Check}(e)$	Check if element e is in BF.
$BF.\mathit{Pos}(e)$	Calculate positions to be changed for element e in BF.
ϵ	BF parameter s.t. the false positive rate is $2^{-\epsilon}$
l	Number of hash functions in the Bloom filter
σ	Symmetric security parameter defining m, the number of base OTs, the size of the exponents
$PRF_k(\cdot)$	Pseudorandom function with secret key k
$AES_k(\cdot)$	AES encryption under secret key k
$\widetilde{AES}_k$	Garbled tables for AES circuit with key k

Table 2. Notation used throughout the paper.

positions for each query, whereas we only need to run a single OPRF instance. [45, 57] provide another construction based on the Goldwasser-Micali homomorphic encryption scheme. However, their protocol reveals several bits of the BF for each query, and clients can learn information from these bits. They improved the Naor-Reingold OPRF-based scheme from [50] using garbled circuits, but still require multiple OPRFs.

Trusted hardware can also be used to instantiate PSI efficiently. The protocol of [30] uses standard smartcards and was extended in [20] to settings where the hardware token(s) are no longer fully trusted. Trusted execution environments such as Intel SGX or ARM TrustZone can be used for PSI as shown in [60].

2 PSI with Precomputation

In this section, we describe the four existing PSI protocols which we experimentally compare in §3. We adapt PSI protocols from the literature to the offline/online setting with online communication linear in the client's smaller set. We give the necessary preliminaries to our protocols in Appendix A, and further on assume that the reader is familiar with the notion of Bloom filters, oblivious transfers and Yao's garbled circuit protocol. Throughout this paper, we use the notations shown in Tab. 2. In three of the four protocols that we describe, the server sends to the client a database of encrypted elements. To reduce the size of the server's encrypted database before transfer, we do not send the raw database of encrypted elements, but rather encode all encrypted elements in a Bloom filter (BF) and send this data structure to the client. We note that the server cannot simply encode its plaintext elements in a BF and send it to the client, since the Bloom filter leaks information about the server's elements [18]. The ciphertexts in the encrypted database are either 128 bit long (AES encryption), 284 or 256 bit long (elliptic curve) or 2048 bit long (finite field for public-key encryption), and therefore using a Bloom filter significantly reduces the size of the transferred and stored data. We note that the usage of a BF introduces potential false positives, but their rate can be controlled (cf. §A.3).

Since our aim is to shift as much communication and computation as possible to the offline phase, we describe the protocols in three phases. Firstly, the *base phase* includes the data-independent precomputation and must be performed in order to setup the underlying primitives within the protocol. The *setup phase* includes the precomputation steps that depend on the elements in the server's database. We note that in three of our protocols the client is not required to do work proportional to the server's set even in this phase, it only receives a Bloom filter in size proportional to the server's set, the size of which is greatly reduced compared to the encrypted database. The *online phase* in all cases includes the query-dependent phase, i.e., the phase where the client's input is required.

Correctness guarantees with our modifications. The correctness guarantees of the modified protocols follow from the correctness of the original protocols in [11, 31, 35, 54] and from the correctness of the Bloom filter (up to false positives). This is due to the fact that the same messages are exchanged between the parties after the same computational steps; the only difference is the order of these messages and the data structure (BF) for storing the encrypted database.

Security guarantees for our modifications. The security guarantees of our protocols follow from the security of the original protocols. The underlying assumptions on which the basic protocols depend are detailed in the respective sections. We apply two modifications to the protocols that do not affect security:

The first modification is *shifting operations into the base and setup phases*, which only means changing the order of messages or operations compared to the original protocols. Since the operations and the communication are the same, the security of the original protocols is not violated when semi-honest adversaries are considered.

The second modification is *replacing sending the server's encrypted elements to the client, with sending a Bloom filter which encodes these elements*. However, sending a BF encoding of a set of values does not reveal any more information than sending the set itself. More precisely, it is possible to show a simple reduction demonstrating that any attack on the modified protocol, which uses a BF, can be changed to an attack on the original protocol which sends the original values: Assume that there is an algorithm A which the client can apply to the modified algorithm (with a BF), and break security with non-negligible probability p. It is easy to devise an algorithm A' which the client can use to break the original protocol. The algorithm A' runs the protocol and feeds all messages that it receives to the algorithm A, with the following change: when it receives the raw set of encrypted elements from the server it first encodes it as a BF and only then feeds it to A. The algorithm A therefore observes the same view as in a run of the modified protocol, and therefore can break security with probability p. This results in A' breaking security with the same probability.

Efficient and secure updates. Our proposed protocols allow efficient updates to the database: Bloom filters suffice for insertion, whereas deletion requires to use counting Bloom filters.

For an *insertion in the Bloom filter*, the server does not need to perform the base and setup phases again and send a whole new BF, it suffices to send the modifications to the BF. This can be done efficiently in either of the following two ways: 1) The server may send the *encrypted element* that needs to be inserted in the BF to the client, or alternatively, 2) the server may send *those positions* of the Bloom filter that need to be changed. For insertion, the bits in the specified positions need to be set to one. Since the elements inserted in the BF are encrypted, the client cannot learn which element was inserted, except when the element is in its set as well. This information, however, leaks also when comparing the results of a PSI run before and one after the change, even if the encrypted database is re-generated by the server using a different key.

A counting Bloom filter (CBF) is an extension of Bloom filters that does not only allow for insertion and lookup, but also for delete operations [19]. Instead of storing bits, the CBF stores small counter values of t bits that are increased by one on insert and decreased by one on delete. In scenarios where deleting elements is of interest (e.g. the messaging application or the malware checking service described in §1), counting Bloom filters can replace BFs in our protocols. Depending on the bit length t of the counters in the Bloom filter, its size becomes $1.44\epsilon N_S t$ bits for N_S elements. *Insertion or deletion in the counting Bloom filter* is the same as the insertion into the BF described above: the server either sends the *encrypted elements* that should be updated or the *positions of the counters* that need to be increased (for insert) or decreased (for delete) by one. Once again, the client can only learn which element was deleted, if the element was in the intersection beforehand (otherwise the client cannot decrypt it). This, however, does not reveal any additional information than what would be revealed by comparing two PSI protocol runs before and after the deletion.

The above mentioned updates are very efficient. The first option that sends the encrypted elements depends on the underlying encryption scheme, i.e., on the size of the ciphertext. The second option depends on the size of the BF or CBF, and the number of hash functions l. These are $l \cdot \log_2(1.44\epsilon N_S)$ bits per element for a BF or $l \cdot \log_2(t \cdot 1.44\epsilon N_S)$ bits per element for a CBF.

Outline. We give the following protocols with our modifications: the RSA blind signature-based protocol (RSA-PSI) in §2.1, the Diffie-Hellman-based PSI protocol (DH-PSI)

$\mathcal{S}$		$\mathcal{C}$
Input: $\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_{N_\mathcal{S}}\}$; Output: $\perp$		Input: $\mathcal{Y} = \{\mathbf{y}_1, ..., \mathbf{y}_{N_\mathcal{C}}\}$; Output: $\mathcal{X} \cap \mathcal{Y}$
$\mathcal{B}, \mathcal{P} := \{\}$	**Base Phase**	$\mathcal{A}, \mathcal{S} := \{\}$
Generate RSA private key d	Agree on ϵ, m-bit RSA modulus N, exponent e	Generate $r_1, \ldots, r_{N_\mathcal{C}^{\max}}$ random
		For $i = 1$ to $N_\mathcal{C}^{\max}$:
		$r_i^{\text{inv}} = r_i^{-1} \mod N$
		$r'_i = (r_i)^e \mod N$
Initialize BF of length $1.44\epsilon N_\mathcal{S}$.	**Setup Phase**	
For $i = 1$ to $N_\mathcal{S}$:		
$BF.\mathit{Insert}((\mathbf{x}_i)^d \mod N)$	BF	
	$\longrightarrow$	
	Online Phase	For $i = 1$ to $N_\mathcal{C}$:
	$\mathcal{A}$	$\mathcal{A}[i] = \mathbf{y}_i \cdot r'_i \mod N$
	$\longleftarrow$	
For $i = 1$ to $N_\mathcal{C}$:		
$\mathcal{B}[i] = (\mathcal{A}[i])^d \mod N$	$\mathcal{B}$	If $BF.\mathit{Check}(\mathcal{B}[i] \cdot r_i^{\text{inv}} \mod N)$ then
	$\longrightarrow$	put $\mathbf{y}_i$ into $\mathcal{S}$
		Output $\mathcal{S}$.
For $i = 1$ to $N_\mathcal{U}$	**Update** $N_\mathcal{U}$ elements	
put $BF.\mathit{Pos}((\mathbf{u}_i)^d \mod N)$ into $\mathcal{P}$	$\mathcal{P}$	
	$\longrightarrow$	Modify BF in positions $\mathcal{P}$

Fig. 1. The RSA Blind Signature based PSI protocol (RSA-PSI).

in §2.2, the Naor-Reingold OPRF-based protocol (NR-PSI) in §2.3, and the OPRF-based protocol using secure evaluation of a garbled AES circuit (GC-PSI) in §2.4.

2.1 RSA Blind Signature-based PSI (RSA-PSI)

A protocol based on RSA blind signature (RSA-PSI) was proposed in [11] and implemented later in [12]. This protocol is a candidate for PSI with unequal number of inputs due to its communication efficiency and its low computation on the client side. The protocol proposed in [11] is such that the client only performs $\mathcal{O}(N_\mathcal{C})$ modular multiplications in the online phase. The originally proposed protocol uses a cryptographic hash function H which we substitute with a Bloom filter BF in order to achieve better communication complexity and lower client storage.

The modified RSA-PSI protocol is depicted in Fig. 1. In the *base phase*, $\mathcal{S}$ and $\mathcal{C}$ agree on the RSA public key (N, e) and the false positive rate for the Bloom filter BF, and $\mathcal{S}$ generates the RSA private key d. In this phase, $\mathcal{C}$ chooses $N_\mathcal{C}^{\max}$ random numbers and calculates their inverses as well as their modular exponentiations to the power e (the RSA public key). Thereafter, in the *setup phase*, $\mathcal{S}$ encrypts its inputs using its private key d and inserts the ciphertexts into the Bloom filter BF initialized before, and sends the BF to $\mathcal{C}$. The *online phase* starts with $\mathcal{C}$ blinding its inputs with the encryption of the respective random values and sending the resulting values to $\mathcal{S}$. $\mathcal{S}$ encrypts these using its private key d and sends the result back to $\mathcal{C}$. $\mathcal{C}$ can then unblind the encrypted blinded values by multiplying each of its inputs with the inverse of the respective random number, due to the property of RSA that $x^{ed} \equiv x \mod N$. Afterwards $\mathcal{C}$ can define the intersection by checking if the unblinded encrypted elements are in the Bloom filter BF that was sent by $\mathcal{S}$ in the setup phase.

Update. For a set of parameters, $\mathcal{S}$ can decide which option for an update is more efficient. In the first option, the server sends 2,048-bit ciphertexts per element to the client. In the second option, $\mathcal{S}$ sends per updated element $l \cdot \log_2(1.44\epsilon N_\mathcal{S})$ bits for a BF, or $l \cdot \log_2(t \cdot 1.44\epsilon N_\mathcal{S})$ bits for a CBF. This means, e.g., 1,063 bits per element for a BF for $N_\mathcal{S} = 2^{30}$ and FPR$= 10^{-9}$ and only 238 bits for $N_\mathcal{S} = 2^{20}$ and FPR$= 10^{-3}$, so for most realistic parameters, this option is more efficient.

Security. The security of the modified RSA-PSI protocol follows from the security of the original protocol described in [11]. As in [11], privacy of both parties is achieved under the RSA assumption which relies on integer factorization being hard. This means that given an m-bit integer N that is the product of two large primes p and q, there exists no polynomial time algorithm to find the two prime factors of N. Different randomness will be used for the inputs of $\mathcal{C}$ if the protocol is run multiple

$\mathcal{S}$		$\mathcal{C}$
Input: $\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_{N_\mathcal{S}}\}$; Output: $\perp$		Input: $\mathcal{Y} = \{\mathbf{y}_1, ..., \mathbf{y}_{N_\mathcal{C}}\}$; Output: $\mathcal{X} \cap \mathcal{Y}$
$\mathcal{A}, \mathcal{C}, \mathcal{P} := \{\}$	**Base Phase**	$\mathcal{B}, \mathcal{S} := \{\}$
Generate secret key α	Agree on ϵ, m-bit prime p	Generate secret key β
Randomly permute elements in $\mathcal{X}$	**Setup Phase**	
For $i = 1$ to $N_\mathcal{S}$:		Initialize BF of length $1.44\epsilon N_\mathcal{S}$
$\mathcal{A}[i] = (\mathbf{x}_i)^\alpha \mod p$	$\mathcal{A}$	For $i = 1$ to $N_\mathcal{S}$:
	$\longrightarrow$	$BF.\mathit{Insert}((\mathcal{A}[i])^\beta \mod p)$
	Online Phase	For $i = 1$ to $N_\mathcal{C}$:
	$\mathcal{B}$	$\mathcal{B}[i] = (\mathbf{y}_i)^\beta \mod p$
For $i = 1$ to $N_\mathcal{C}$:	$\longleftarrow$	
$\mathcal{C}[i] = (\mathcal{B}[i])^\alpha \mod p$	$\mathcal{C}$	If $BF.\mathit{Check}(\mathcal{C}[i])$ then
	$\longrightarrow$	put $\mathbf{y}_i$ into $\mathcal{S}$.
		Output $\mathcal{S}$.
For $i = 1$ to $N_\mathcal{U}$	**Update** $N_\mathcal{U}$ elements	
put $BF.\mathit{Pos}((\mathbf{u}_i)^d \mod p)$ into $\mathcal{P}$	$\mathcal{P}$	
	$\longrightarrow$	Modify BF in positions $\mathcal{P}$

Fig. 2. The Diffie-Hellman-based PSI protocol (DH-PSI).

times, and therefore, unlinkability of the client's inputs can also be achieved.

Online communication. In the online phase, $\mathcal{C}$ sends its blinded inputs to $\mathcal{S}$, which are $mN_\mathcal{C}$ bits, where m is the bit length of N. $\mathcal{S}$ responds with $mN_\mathcal{C}$ bits, which yields a total of $2mN_\mathcal{C}$ bits communication.

Online computation of $\mathcal{C}$. The online computation performed by the computationally restricted party $\mathcal{C}$ is $2N_\mathcal{C}$ efficient modular multiplications and $N_\mathcal{C}$ checks if an element is in the Bloom filter or not. This is performed by $lN_\mathcal{C}$ hash function evaluations.

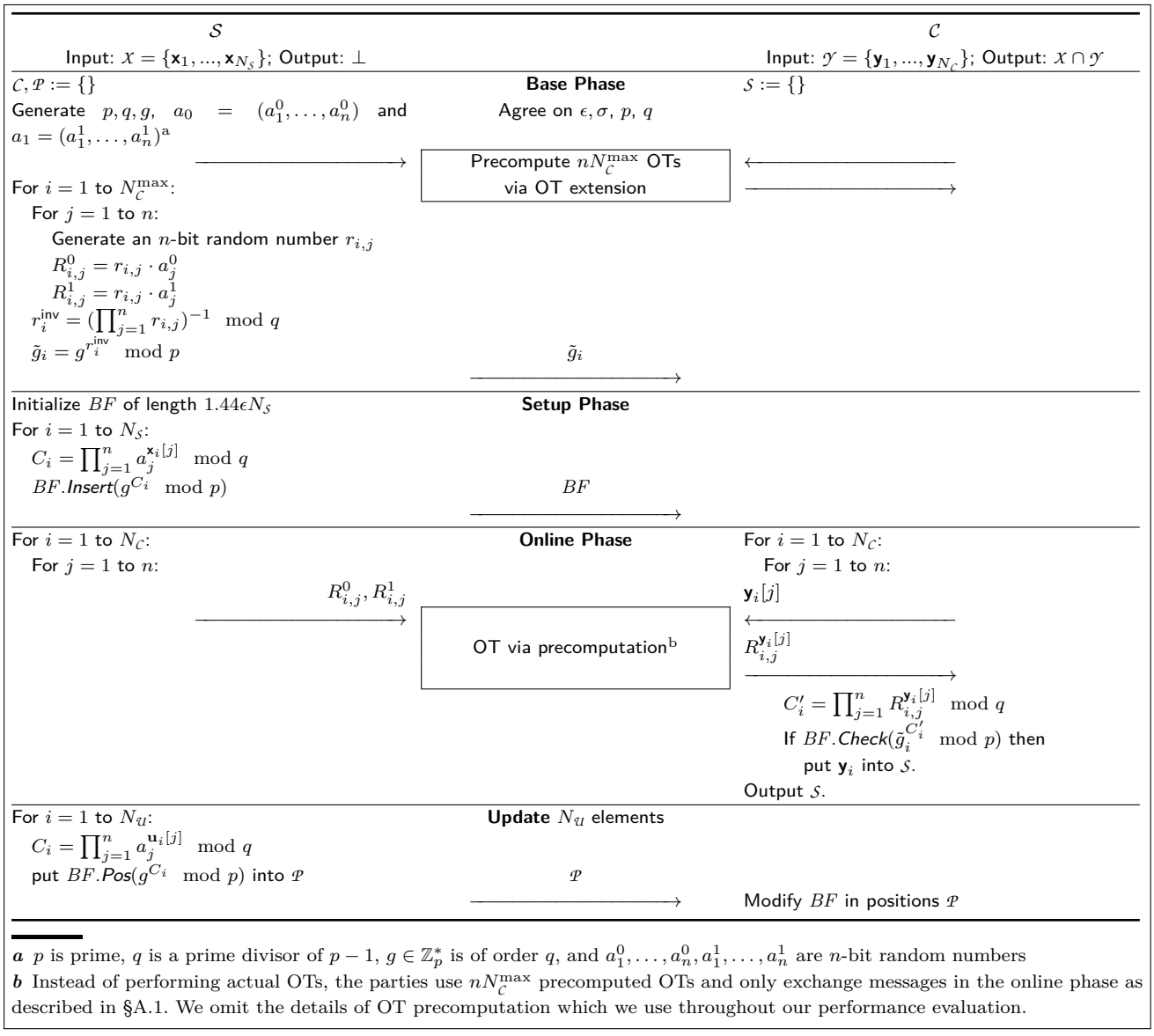
2.2 Diffie-Hellman-based PSI (DH-PSI)

The first PSI protocol with linear communication complexity, based on the Diffie-Hellmann protocol [15], was proposed in [35] (DH-PSI). It was proposed for a scenario where private preferences of two parties are matched using a known preference-matching function. The Diffie-Hellman-based PSI protocol has previously been used in [44] for a matchmaking protocol where users verify their matching credentials without revealing them to each other or to a trusted third party. This protocol can be implemented based on elliptic-curve cryptography and has linear communication complexity.

The modified DH-based PSI protocol, which places the encryptions into Bloom filters instead of using hash functions, is shown in Fig. 2. In a nutshell, the protocol starts by the parties agreeing on a cyclic group of prime order p and a parameter ϵ for the false positive rate of the Bloom filter, and generating their secret exponents in the *base phase*. Later in the *setup phase*, the server $\mathcal{S}$ computes the encryptions of its inputs by raising them to its secret exponent, and sends them over to $\mathcal{C}$. $\mathcal{C}$ initializes the Bloom filter BF, into which $\mathcal{C}$ inserts the encryptions of the received values with its secret exponent. We note that this protocol is the only studied protocol where the client needs to generate the Bloom filter and cannot receive it from the server directly. The *online phase* consists of the client $\mathcal{C}$ first encrypting its input with its secret key and sending it over to $\mathcal{S}$. $\mathcal{S}$ then raises the received values to its own exponent and sends back the results to the client $\mathcal{C}$. In our scenario only the client $\mathcal{C}$ learns the output of the PSI protocol, by checking if its encrypted inputs are in the Bloom filter.

Update. The efficiency of an update for DH-PSI is the same as for RSA-PSI (cf. §2.1), except when implemented using elliptic curve cryptography. E.g., the popular curve P-256 for 128-bit security has 256-bit ciphertexts and therefore, in many cases the first option (sending the encrypted elements) is more efficient.

Security. The security of the modified Diffie-Hellman based PSI (DH-PSI) protocol follows from the security of the protocol described in [35]. The privacy of the inputs of both $\mathcal{S}$ and $\mathcal{C}$ is achieved under the decisional Diffie-Hellman hardness assumption. This means that in a cyclic group G with generator g, for uniformly random elements α, β, γ, it is impossible to distinguish $(g^\alpha, g^\beta, g^{\alpha\beta})$ from $(g^\alpha, g^\beta, g^\gamma)$. Note that if $\mathcal{C}$ queries the same item twice, $\mathcal{S}$ will notice this fact. To remedy this, $\mathcal{C}$ needs to generate a new secret key for each protocol run. However, in this case, $\mathcal{C}$ can no longer use a Bloom filter to save storage.



a p is prime, q is a prime divisor of $p-1$, $g \in \mathbb{Z}_p^*$ is of order q, and $a_1^0, \ldots, a_n^0, a_1^1, \ldots, a_n^1$ are n-bit random numbers

b Instead of performing actual OTs, the parties use $nN_\mathcal{C}^{\max}$ precomputed OTs and only exchange messages in the online phase as described in §A.1. We omit the details of OT precomputation which we use throughout our performance evaluation.

Fig. 3. Naor-Reingold PRF-based protocol (NR-PSI).

Online communication. In the online phase, $\mathcal{C}$ sends its encrypted inputs to $\mathcal{S}$, with a total of $mN_\mathcal{C}$ bits, where m is the length of a group element. Afterwards, $\mathcal{S}$ sends back $mN_\mathcal{C}$ bits to $\mathcal{C}$. Therefore, the total online communication of the protocol is $2mN_\mathcal{C}$ bits.

Online computation of $\mathcal{C}$. In the online phase, the client needs to compute $N_\mathcal{C}$ modular exponentiations and check $N_\mathcal{C}$ elements in the Bloom filter, which requires $lN_\mathcal{C}$ hash function evaluations.

2.3 Naor-Reingold PRF-based PSI (NR-PSI)

A PSI protocol based on the Naor-Reingold pseudo-random function (PRF) [48] (NR-PSI) was proposed in [31]. The idea of evaluating PRFs in an oblivious manner was presented in [23] where its application to PSI was first mentioned and later studied in [37]. The plain computation of the Naor-Reingold PRF is very efficient, $\text{PRF}_a(x)$ can be computed with one modular exponentiation and n modular multiplications.

The protocol shown in Fig. 3 works as follows: in the *base phase*, the parties agree on all parameters: a prime number p, a prime divisor of $p-1$ denoted by q, an

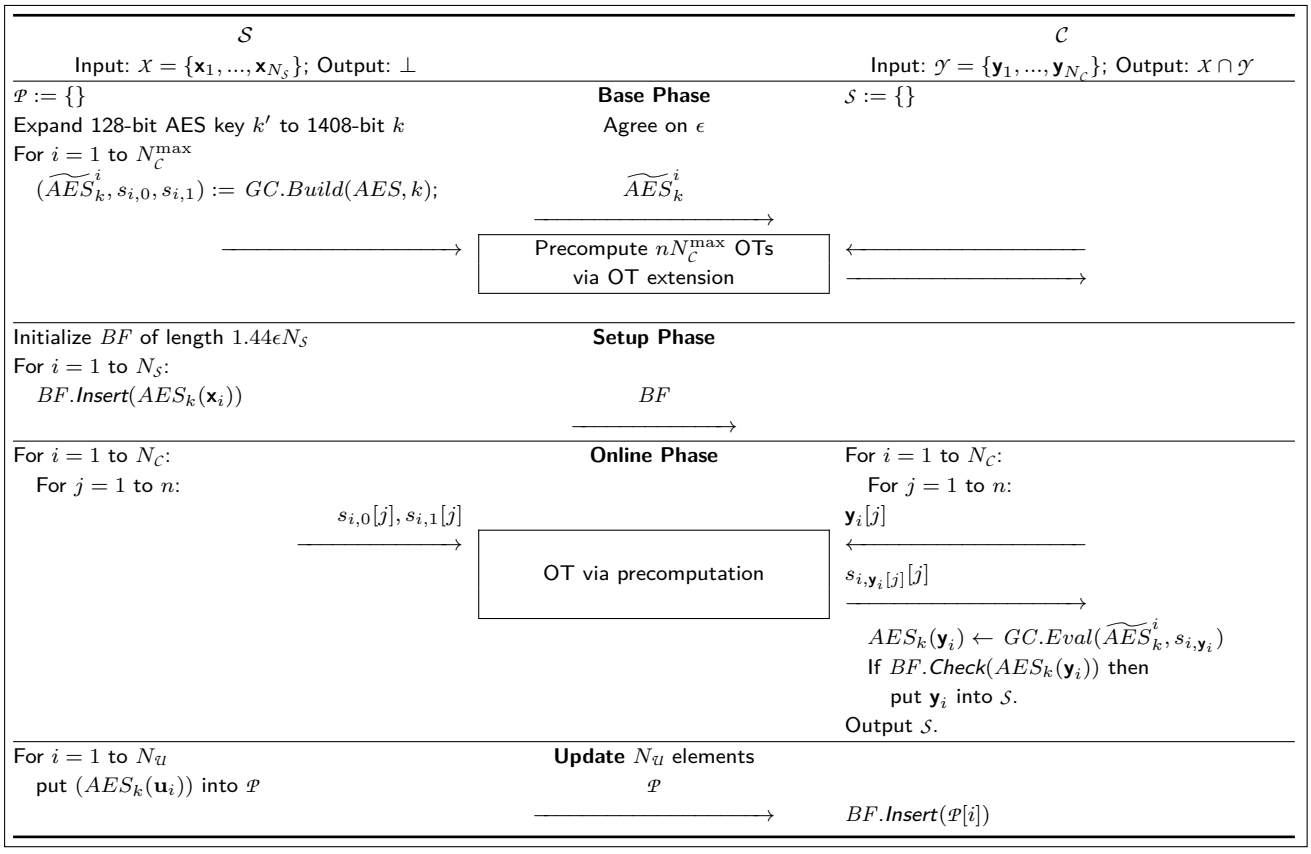


Fig. 4. The AES GC-based PSI protocol (GC-PSI).

element $g \in \mathbb{Z}_p^*$ of multiplicative order q, ϵ the FPR parameter of BF and σ the symmetric security parameter. Then $N_\mathcal{C}^{\max}$ random numbers are generated, and each one is split into the multiplication of n random values. Inverses of all the $N_\mathcal{C}^{\max}$ random numbers are calculated by $\mathcal{S}$. In this phase, $nN_\mathcal{C}^{\max}$ oblivious transfers are precomputed using OT extension, for which σ base OTs are computed as a first step. Thereafter, in the data dependent *setup phase*, the server $\mathcal{S}$ encrypts its elements with the Naor-Reingold PRF using the key generated before. $\mathcal{S}$ inserts these values into the BF and sends it to $\mathcal{C}$. In the *online phase*, for each bit of each element of $\mathcal{C}$, the parties exchange the necessary information in order for the client $\mathcal{C}$ to recover the exponents for calculating the Naor-Reingold PRF evaluation of its inputs. This is done efficiently by making use of the precomputed OTs. Then, by checking if its encrypted elements are in the BF or not, $\mathcal{C}$ is able to define the set intersection.

Update. The efficiency of an update for NR-PSI is the same as for RSA-PSI (cf. §2.1), i.e., the second option (sending positions) should be preferred.

Security. The security of the modified protocol based on the Naor-Reingold OPRF (NR-PSI) is based on the security guarantees of [31]. Its security is proven under the decisional Diffie-Hellman assumption (cf. §2.2). When considering security against semi-honest adversaries, the OT extension protocol of [2] can be used in the base phase of NR-PSI.

Online communication. The online communication in NR-PSI consists of the client sending $nN_\mathcal{C}$ bits to the server, who then responds with $2nmN_\mathcal{C}$ bits and additionally $nN_\mathcal{C}$ bits. This adds up to a total communication of $2nN_\mathcal{C}(1+m)$ bits.

Online computation of $\mathcal{C}$. In this protocol, the client performs $nN_\mathcal{C}(1+m)$ XOR operations for the precomputed OTs, $nN_\mathcal{C}$ modular multiplications and $N_\mathcal{C}$ modular exponentiation for evaluating the PRF and $lN_\mathcal{C}$ hash function evaluations for the Bloom filter check.

2.4 AES GC-based PSI (GC-PSI)

OPRF-based PSI can also be achieved by evaluating AES with Yao's garbled circuits protocol as proposed in [54]. This protocol is shown in Fig. 4. The *base phase* starts with the server $\mathcal{S}$ generating and expanding the

secret key for AES, retrieving secret key k'. Then, $N_{\mathcal{C}}^{\max}$ garbled AES circuits are generated for each elements of $\mathcal{C}$, where $N_{\mathcal{C}}^{\max} \geq N_{\mathcal{C}}$ is the maximal number of inputs of $\mathcal{C}$. These garbled AES circuits are then sent to $\mathcal{C}$. Besides this, $nN_{\mathcal{C}}^{\max}$ OTs are precomputed using OT extension, where for random choice bits of $\mathcal{C}$, a message from a random message pair of $\mathcal{S}$ is received by $\mathcal{C}$. Then, the *setup phase* consists of the server $\mathcal{S}$ inserting the AES encryptions of its $N_{\mathcal{S}}$ elements to a BF and sending it to the client $\mathcal{C}$. In the *online phase*, $\mathcal{C}$ obliviously retrieves the garbled Yao keys corresponding to its input bits using the precomputed OTs. Then, $\mathcal{C}$ evaluates the garbled AES circuits on each of its garbled inputs and checks if the resulting value is in the Bloom filter sent by $\mathcal{S}$, which yields the intersection of the sets.

An AES circuit using the S-box of [9] has 5,120 AND gates [32]. Alternatively, we could use the LowMC cipher of [1] with only 2,268 AND gates. Due to recent attacks on LowMC [16, 17], we choose to instantiate the PRF with AES. Recently, [29] proposed "MPC-friendly" PRFs that might provide more efficient primitives for OPRF-based PSI. We leave the examination of these PRFs as future work. Note that the AES key may need to be changed after some time. In this case, the BF must be rebuilt and the garbled circuits need to be regenerated.

Update. For GC-PSI, sending the 128 bit ciphertexts per updated element is more efficient already for $N_{\mathcal{S}} = 2^{15}$ and FPR$= 10^{-3}$, and therefore this first solution should always be preferred.

Security. The security of the GC-PSI protocol depicted in Fig. 4 relies on the well-established assumption that AES is a PRF and on the security of Yao's garbled circuits protocol as proven in [42]. As for NR-PSI, for security against semi-honest adversaries we use the semi-honest OT extension protocol of [2].

Online communication. The online communication is defined by the $nN_{\mathcal{C}}$ bits sent by $\mathcal{C}$ to $\mathcal{S}$ and the $2nmN_{\mathcal{C}}$ bits sent back by $\mathcal{S}$. This means altogether $nN_{\mathcal{C}}(1+2m)$ bits of online communication.

Online computation of $\mathcal{C}$. The client needs to compute $nN_{\mathcal{C}}$ bit-XOR operations, evaluate $N_{\mathcal{C}}$ garbled circuits and check if the encryption of its elements are in the Bloom filter BF. For evaluating a garbled circuit, due to the half-gates optimization of [63] and fixed-key AES garbling of [7], two fixed-key AES evaluations need to be performed per AND gate.

3 Performance Evaluation

We have implemented all four protocols and systematically compare their performance in this section. Our open-source implementation can be found at http://encrypto.de/code/MobilePSI.

Our performance results are given in Tab. 3 for the PC setting and in Tab. 4 for the smartphone setting. The communication and minimal necessary client storage are given in Tab. 5 (see Tab. 6 in Appendix B for the formulas used to compute these). We detail our experiments next.

Parameters. We assume the inputs are 128-bit strings (i.e., $n = 128$). To provide assessment for a sufficient security level, we set our symmetric security parameter σ to 128 and follow the recommendations of [27]. Therefore, where oblivious transfer is used, we perform 128 base OTs for OT extension. In the PC setting, we used the semi-honest OT extension implementation of [2] and in the smartphone setting, we used the OT extension protocol of [36]. This affects only the performance of the base phase since we use precomputed OTs as described in §A.1. For all protocols, we set the length of the modulus as $m = 2{,}048$ bits (284 bits for ECC with K-283 on PC and 256 bits for ECC with P-256 on smartphone) and use a small RSA public exponent e, while for DH-PSI and NR-PSI, we use 256-bit exponents to achieve the desired level of security [27].

Following the advices of a major AV vendor, we perform experiments with realistic parameter choices in §3.1. Furthermore, we elaborate on the behavior of the protocols when varying parameters in §3.2.

Environment. We ran our benchmarks in two different settings, using x86-based PCs and an ARM-based smartphone. Each reported result is an average of 10 runs. In our experiments, we do not make use of parallelization techniques, which could further improve the performance.

In the *PC setting*, the protocols were implemented in C/C++ and ran between two desktops, with Intel Core i7 with 3.5 GHz and 16 GB RAM. Our experiments were performed in LAN and Wifi settings, both of which are with 1 Gbps switches. We used the GNU multiple precision arithmetic library [21] for the cryptographic operations, the ABY framework [14] to generate and evaluate the garbled circuits efficiently, and the BF implementation of [52]. The optimized AES circuit has 5,440 AND gates. For ECC-DH-PSI, we modify the implementation of [55], which uses Koblitz curve K-283

for 128-bit security, where the bit size of the points is $m = 284$.

In the *smartphone setting*, the server-side program ran on a laptop with Intel Core i7 with 4 cores at 2.2 GHz, and the client-side program ran on an Android 6.0.1 phone (Samsung Galaxy S5) with Quad-core 2.5 GHz Krait 400. The two devices were connected via a Wifi connection. We used Java for the Android programming, Bouncy Castle APIs [8] for the cryptographic operations, and FlexSC [43] as the garbled circuit backend. In the following text, we put the results from the smartphone setting in brackets. For ECC-DH-PSI, we used X9.62/SECG curve P-256 for 128-bit security, where the bit size of the points is $m = 256$.

3.1 Experimental Evaluation with Realistic Parameters

We set the maximum number of client inputs to $N_{\mathcal{C}}^{\max} = 1{,}024$ to measure the base phase and the number of server inputs to $N_{\mathcal{S}} = 2^{20}$ to measure the setup phase. We benchmark false positive rates (FPRs) 10^{-3} and 10^{-9} to measure the time usage to build and send a BF. If the application allows for false positives, e.g., the malware checking example from §1, a larger FPR leads to better performance. For more sensitive applications such as the search in chemical compound database, only one false positive in a billion elements should be allowed, i.e., FPR= 10^{-9}. In the online phase, we perform our experiments with $N_{\mathcal{C}} = 1$ to retrieve the performance measurements for each of $\mathcal{C}$'s inputs. Since the online phase of all protocols scales linearly in $N_{\mathcal{C}}$, these numbers can be used to estimate the performance with larger $N_{\mathcal{C}}$ (cf. §3.2).

Base Phase. In the base phase of RSA-PSI, $\mathcal{C}$ needs to generate $N_{\mathcal{C}}^{\max}$ random numbers, and calculate their inverse and modular exponentiation with the public exponent. It takes 56 ms (5,492 ms) for a maximum of $N_{\mathcal{C}}^{\max} = 2^{10}$ client inputs. In DH-PSI, $\mathcal{S}$ and $\mathcal{C}$ do not need to do anything other than transferring a prime number p. In the base phase of NR-PSI and GC-PSI, $\mathcal{S}$ and $\mathcal{C}$ need 82 ms (1,003 ms) to run 128 base OTs, and they also need 31 ms (44,032 ms) to run $128 \cdot N_{\mathcal{C}}^{\max}$ OT extensions. In addition, NR-PSI requires $\mathcal{S}$ to generate $N_{\mathcal{C}}^{\max}$ 128-bit random numbers, multiply them to the secrets, and also calculate their inverse. GC-PSI requires $\mathcal{S}$ to generate and transfer $N_{\mathcal{C}}^{\max}$ garbled circuits in advance, which takes 1,199 ms (411,648 ms).

The communication for NR-PSI is defined by the 128 base OTs and the $128 \cdot N_{\mathcal{C}}^{\max}$ OTs which means transferring around 4 Mbytes, whereas in GC-PSI, $\mathcal{S}$ additionally sends 1,024 garbled circuits to $\mathcal{C}$ which add up to around 178 Mbytes. This additionally has to be stored on the client's side as well, requiring a large storage capacity when large $N_{\mathcal{C}}^{\max}$ is considered.

Setup Phase. In the setup phase, $\mathcal{S}$ needs to encrypt $N_{\mathcal{S}} = 2^{20}$ entries. The GC-PSI approach is far more efficient than the other three schemes in this phase, since it only requires $\mathcal{S}$ to do efficient AES encryption on each entry. However, all the other three schemes require expensive public-key operations on each entry. Our experiments validate this. Due to the hardware-accelerated AES using AES-NI, GC-PSI only takes 70 ms (1,120 ms without AES-NI in Java)[2] to encrypt $N_{\mathcal{S}} = 2^{20}$ entries, while the other three schemes take several minutes to encrypt the same number of entries.

The time usage to build and send a Bloom filter is the same for RSA-PSI, DH-PSI, and NR-PSI, which takes 309 ms (1,590 ms) for $N_{\mathcal{S}} = 2^{20}$ entries and 10^{-3} FPR, and 767 ms (7,364 ms) for the same number of entries but 10^{-9} FPR. The exception is DH-PSI which requires $\mathcal{S}$ to transfer $N_{\mathcal{S}} = 2^{20}$ ciphertexts to $\mathcal{C}$ instead of the BF. Furthermore, it is more expensive to build a Bloom filter on the Android platform, i.e., 15,365 ms for a FPR of 10^{-3} and 46,998 ms for a FPR of 10^{-9}.

In RSA-PSI, NR-PSI and GC-PSI, the communication in the setup phase is defined by the Bloom filter created for $N_{\mathcal{S}} = 2^{20}$ server inputs with 10^{-3} and 10^{-9} FPRs, and is therefore the same for all the three protocols, i.e., around 1.8 Mbytes and 5.4 Mbytes for the respective FPR rates. However, the communication for DH-PSI and ECC-DH-PSI is independent from the FPR of the BF and is around 256 Mbytes and 36 Mbytes, respectively, for $N_{\mathcal{S}} = 2^{20}$ server inputs. For DH-PSI, however, the received values can be processed one by one and inserted to the BF, and therefore the required storage is similar for all four schemes in this phase.

Online Phase. The online phase is measured from the time that $\mathcal{C}$ sends a query until it gets the answer. The query time for GC-PSI on Android is long (8,470 ms) since the garbled circuit evaluation is expensive on a smartphone. However, the garbled circuit evaluation time is significantly improved by AES-NI on PC. If we do the same garbled circuit evaluation on a

2 We note here that our server-side implementation for the smartphone setting is also in Java. The setup phases of the protocols can be as efficient as shown in Tab. 3, except for the setup phase of DH-PSI and ECC-DH-PSI, where the bottleneck is the computation on the client side.

	Base (ms)	Setup (ms)			Online (ms)	Update (ms)	
Task		Encryption	BF				
FPR		-	10^{-3}	10^{-9}	-	10^{-3}	10^{-9}
RSA-PSI (Fig. 1)	56	3,441,906	309	767	7.38	0.11	0.15
DH-PSI (Fig. 2)	**1**	462,496	**257**	**648**	3.49	0.11	0.15
ECC-DH-PSI (Fig. 2)	**1**	1,325,400	**257**	**648**	2.91	0.11	0.15
NR-PSI (Fig. 3)	119	758,400	309	767	10.82	0.11	0.15
GC-PSI (Fig. 4)	1,312	**70**	309	767	**2.49**	**0.09**	**0.09**

Table 3. Runtimes in milliseconds in the **PC setting** in LAN setting with AES-NI. The parameter choices are as follows: $N_S = 2^{20}$, $N_C^{\max} = 2^{10}$, $N_C = 1$, $\sigma = 128$, $n = 128$, $m = 2{,}048$ (284 for ECC), and False Positive Rates (FPRs) of 10^{-3} and 10^{-9}. Best values marked in bold.

	Base (ms)	Setup (ms)			Online (ms)	Update (ms)	
Task		Encryption	BF				
FPR		-	10^{-3}	10^{-9}	-	10^{-3}	10^{-9}
RSA-PSI (Fig. 1)	5,492	19,892,745	**1,590**	**7,364**	60	8	11
DH-PSI (Fig. 2)	**1**	3,014,656	50,880	172,896	**23**	8	11
ECC-DH-PSI (Fig. 2)	**1**	167,837,696	50,880	172,896	363	8	11
NR-PSI (Fig. 3)	45,035	12,100,105	**1,590**	**7,364**	247	8	11
GC-PSI (Fig. 4)	456,683	**1,851**	**1,590**	**7,364**	8,470	**4**	**4**

Table 4. Runtimes in milliseconds in the **smartphone setting** with Wifi connection. The parameter choices are as follows: $N_S = 2^{20}$, $N_C^{\max} = 2^{10}$, $N_C = 1$, $n = 128$, $m = 2{,}048$ for RSA-PSI, DH-PSI, NR-PSI, 284 for ECC-DH-PSI and 128 for GC-PSI, and False Positive Rates (FPRs) of 10^{-3} and 10^{-9}. Best values marked in bold.

	Base (KB)	Setup (KB)		Online (KB)	Client Storage (KB)		Update (KB)	
FPR	-	10^{-3}	10^{-9}	-	10^{-3}	10^{-9}	10^{-3}	10^{-9}
RSA-PSI (Fig. 1)	**0**	**1,840**	**5,521**	0.5	2,353	6,034	0.029	0.031
DH-PSI (Fig. 2)	**0**	262,144	262,144	0.5	1,841	**5,521**	0.029	0.031
ECC-DH-PSI (Fig. 2)	**0**	36,352	36,352	**0.1**	**1,840**	**5,521**	0.029	0.031
NR-PSI (Fig. 3)	4,201	**1,840**	**5,521**	4.0	8,390	7,585	0.029	0.031
GC-PSI (Fig. 4)	181,482	**1,840**	**5,521**	4.0	179,136	182,817	**0.016**	**0.016**

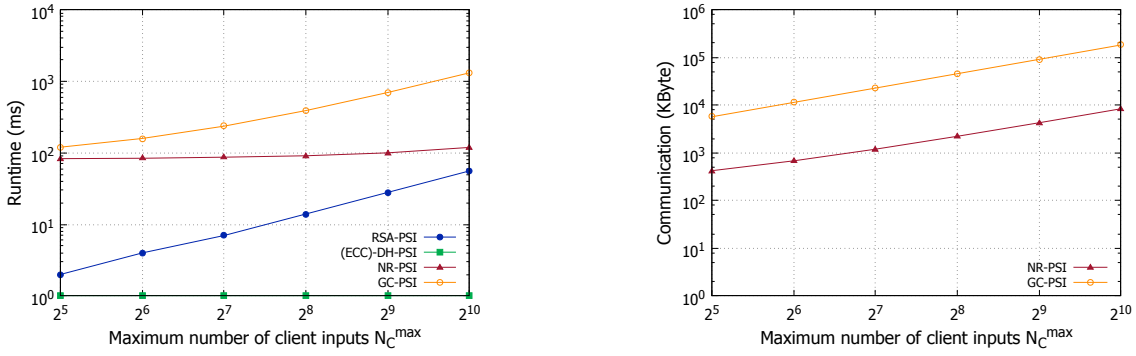
Table 5. Communication in Kilobytes in the different protocols and the storage capacity required by $\mathcal{C}$. The parameter choices are as follows: $N_S = 2^{20}$, $N_C^{\max} = 2^{10}$, $N_C = 1$, $n = 128$, $m = 2{,}048$ for RSA-PSI, DH-PSI, NR-PSI, 284 for ECC-DH-PSI on PC and 128 for GC-PSI, and False Positive Rates (FPRs) of 10^{-3} and 10^{-9}. Best values marked in bold.

machine that supports AES-NI, it takes only 2.49 ms, which is in the same order of magnitude as the other three schemes. Furthermore, in the PC setting, GC-PSI is the most efficient solution in the online phase, requiring the least expensive operations offline as well. In the smartphone setting, due to the OTs needed in NR-PSI and GC-PSI, DH-PSI performs best in the online phase, providing a query response in just 23 ms. This shows the practicality of our approach: for the maximal number of client inputs $N_C = N_C^{\max} = 2^{10}$, our best approach has an online runtime of only 2.55 seconds (23.55 seconds).

The communication in the online phase depends only on the client's inputs. RSA-PSI and DH-PSI need to transfer only 0.5 Kbytes per query for two group elements, while NR-PSI and GC-PSI need to communicate via the precomputed OTs. This requires around 4 Kbytes of data transfer per query.

Update. As we discussed in §2, updates can be done in two ways: 1) $\mathcal{S}$ sends the encryptions of the newly added elements, or 2) $\mathcal{S}$ sends the positions that need to be changed. Based on our parameters, we chose the first solution for GC-PSI and the second solution for the other protocols. The rationale is that, for an FPR of either 10^{-3} or 10^{-9}, $l \cdot \log_2(1.44\epsilon N_S)$ is larger than the size of a symmetric cipher (i.e., 128-bit), but smaller than a asymmetric cipher (i.e., 2048-bit). The results show that updates can be done efficiently for all four protocols.

Summary. In conclusion, our experiments show that DH-PSI has the most efficient base phase since it



(a) Runtime (ms) **(b)** Communication (KBytes)

Fig. 5. Effect of varying maximum client input sizes in the base phase for $N_C^{\max} = 2^i, i \in \{5, \ldots, 10\}$.

only requires $\mathcal{S}$ and $\mathcal{C}$ to generate their secret keys. GC-PSI has the fastest setup phase, especially when AES-NI is supported. For the online phase, GC-PSI and DH-PSI are the most efficient solutions in the PC setting and smartphone setting, respectively. The protocols requiring the highest communication are GC-PSI and DH-PSI due to the garbled circuits and the transferred ciphertexts, respectively. The lowest communication and storage overhead is observed in RSA-PSI and NR-PSI.

3.2 Experimental Evaluation of Varying Parameters

In this section, we discuss the effect of varying certain parameters. We show that linear scaling can be observed when varying the parameters N_S, N_C, $N_C^{\max}$ and the false positive rate (FPR). The runtime experiments in this section were performed in the *PC setting* in LAN as before, with the same security parameter $\sigma = 128$. We note that the figures in this section are depicted in logarithmic scale in both axes.

3.2.1 Base Phase

The base phase of the protocols performs precomputation that is independent of the parties' inputs. These operations are: key generation for RSA-PSI and DH-PSI, the generation of $N_C^{\max}$ blinded random values in RSA-PSI, OT precomputation in NR-PSI and GC-PSI, GC precomputation in GC-PSI, and the generation and transfer of $N_C^{\max}$ group elements in NR-PSI.

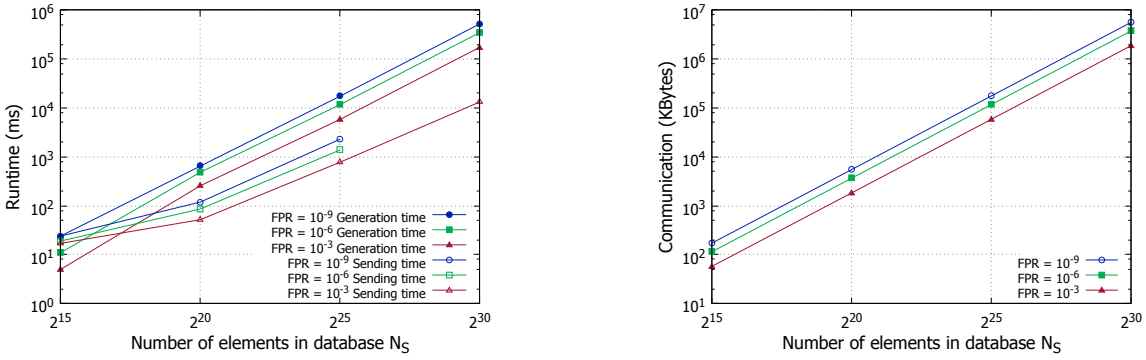
Varying Maximum Client Input Size.
This phase is only affected by the choice of the maximum number of client inputs, as shown in Fig. 5 for $N_C^{\max}$ between $2^5 = 32$ and $2^{10} = 1{,}024$.

Runtimes (Fig. 5a). It can be observed that the most computationally heavy protocol in the base phase is GC-PSI due to the precomputation of $N_C^{\max}$ garbled AES circuits, which requires up to 13.1 seconds for $N_C^{\max} = 1{,}024$. The work performed in the base phase of NR-PSI is mainly dominated by the base OTs which cost 82 ms. For RSA-PSI, $N_C^{\max}$ modular inverses and exponentiations with a small exponent (public key) are computed, the cost of which scale linearly in $N_C^{\max}$, and reach 56 ms for $N_C^{\max} = 1{,}024$. DH-PSI is independent of $N_C^{\max}$ as only the secret keys are initialized.

Communication (Fig. 5b). The only communication in NR-PSI and GC-PSI is the OT precomputation and transferring $N_C^{\max}$ group elements or $N_C^{\max}$ garbled circuits, respectively. The most communication happens in the case of GC-PSI, where for 1,024 elements, 178 MBytes of garbled circuits need to be transferred and stored by the client. However, in applications where $N_C^{\max}$ is an order of magnitude smaller, e.g., for $N_C^{\max} = 2^6 = 64$, GC-PSI only uses 11 MBytes, which is not prohibitive for a smartphone either. NR-PSI requires 8.1 MBytes of communication for $N_C^{\max} = 1{,}024$ and 680 KBytes for $N_C^{\max} = 2^6 = 64$. RSA-PSI and DH-PSI do not require communication in this phase.

3.2.2 Setup Phase

In the setup phase, computation and communication that depend on the server's input elements are performed. The following operations are performed: in RSA-PSI, NR-PSI and GC-PSI, the server encrypts its



(a) Runtime (ms) **(b)** Communication (KBytes)

Fig. 6. Bloom filter related effect of varying database sizes for $N_S = 2^i, i \in \{15, 20, 25, 30\}$ and false positive rates for FPR$= 10^{-i}, i \in \{3, 6, 9\}$ in the setup phase.

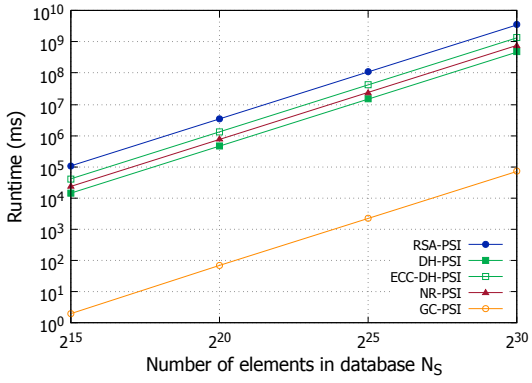


Fig. 7. Effect of varying database sizes for $N_S = 2^i, i \in \{15, 20, 25, 30\}$ in the setup phase, independent from Bloom filter.

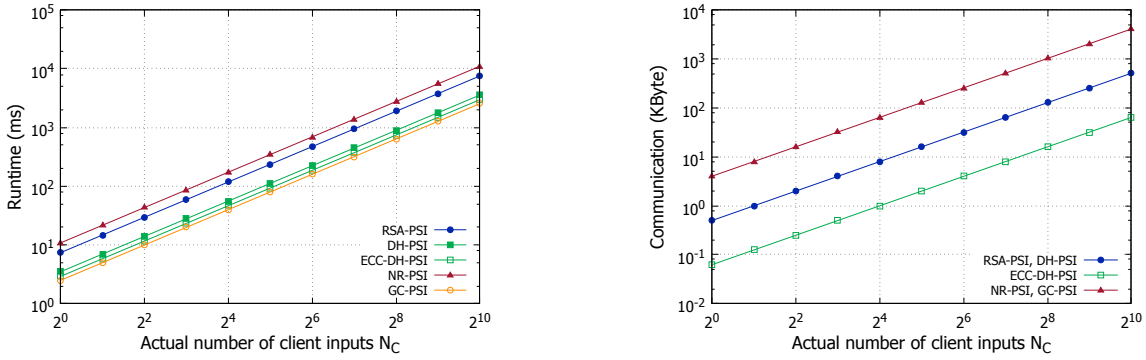
database of size N_S and inserts the encrypted elements into a Bloom filter of size $1.44\epsilon N_S$ that is sent to the client. In DH-PSI, the client generates the Bloom filter after receiving the encrypted elements from the server. ϵ is chosen in all case as a Bloom filter parameter such that the false positive rate in the BF is $2^{-\epsilon}$. The size of the server's set N_S and ϵ determine the efficiency of this phase, which depends on the underlying application scenario.

Varying Database Size and False Positive Rate - Effect on Bloom Filter.

The size of the database N_S and the parameter for the false positive rate ϵ affect the size of the Bloom filter and therefore the efficiency of the setup phase. Moreover, ϵ determines the number of hash functions used in the BF that has impact on the runtime in the online phase. This cost, however, is negligible and therefore, we show experimental results for the time generation and size of the BF for false positive rates 10^{-3}, 10^{-6} and 10^{-9} and varying the size of the database N_S between 2^{15} and 2^{30}. Fig. 6 shows the effect on the runtime and the communication in the setup phase.

Runtimes (Fig. 6a). We can see that the growth in both runtime and communication is linear in the size of the server's set N_S. We separate the runtime used for generating and that used for sending the BF from the server to the client for two reasons: firstly, this second cost is not present in DH-PSI, where all encrypted elements are sent instead of the BF, and a BF is locally generated by the client. Secondly, the network setting can affect the performance when it comes to sending the BF, but the generation in that case will stay constant.

Communication (Fig. 6b). We can observe that the required storage capacity is not prohibitive even on a smartphone, since e.g., for $N_S = 2^{20}$ elements (depending on FPR) the client needs between 1.8 and 5.4 MBytes of storage. For applications on larger databases, e.g., $N_S = 2^{25}$, the communication and storage required is between 57.5 and 172.5 Mbytes, which is not prohibitive in the PC setting. The generation of



(a) Runtime (ms) **(b)** Communication (KBytes)

Fig. 8. Effect of varying client input sizes in the online phase for $N_C = 2^i, i \in \{0, 1, \ldots, 10\}$.

such a large database can take between 5.8 and 17.4 seconds, while sending these over a LAN network takes 0.8 to 2.3 seconds.

Varying Database Size - Effect on Runtime. In RSA-PSI and NR-PSI, the server performs N_S expensive public-key encryptions in the setup phase. In DH-PSI, both parties perform a public-key operation per element in the database and the server has to transfer N_S group elements to the client. When using GC-PSI, the server performs only N_S symmetric-key operations.

Runtimes (Fig. 7). The gap between public-key operations and symmetric encryption is clearly visible and provides an advantage to GC-PSI (which, in turn, requires large storage capacity as mentioned before). $N_S = 2^{25}$ elements in the setup phase of RSA-PSI (where both the modulus and the exponent have 2,048 bits), are encrypted in around 30 hours. For NR-PSI and DH-PSI this takes only around 4 hours, while GC-PSI terminates within 2.2 seconds. We note here that the only protocol affected in this benchmark by the network setting is DH-PSI, where the ciphertexts need to be exchanged between the two parties.

3.2.3 Online Phase

The online phase is only affected by the number of inputs in the client's set. Fig. 8 summarizes the performance of our protocols.

Varying Client Input Size. For all inputs of the client, RSA-PSI and DH-PSI requires public-key operations to be performed, NR-PSI requires an oblivious PRF evaluation with precomputed OT, and GC-PSI requires an AES garbled circuit evaluation.

Runtimes (Fig. 8a). We vary the number of client input N_C from 1 to 1,024. For a realistic choice of $N_C = 2^6 = 64$, all protocols terminate in less than a second. GC-PSI provides the best performance of 159 ms. For the maximum number of inputs $N_C = 2^{10} = 1{,}024$, the protocols require 11 minutes in the worst case (NR-PSI) and 2.5 minutes in the best case (GC-PSI).

Communication (Fig. 8b). In RSA-PSI and DH-PSI, the parties exchange two 2,048-bit messages for each element in the client's set, resulting in at most 512 KBytes of data transfer for $N_C = 2^{10} = 1{,}024$. NR-PSI and GC-PSI use precomputed OTs for each bit of the client's inputs and therefore require more communication, around 4 KBytes per element., For 1,024 elements this yields around 4 MBytes of communication.

4 Extensions

In this section, we propose two possible extensions for the four PSI protocols detailed in §2. Three further extensions are discussed in Appendix C.

4.1 Same Encrypted Database for Multiple Clients for RSA-PSI and DH-PSI

In our motivating scenarios in §1, the encrypted database sent by the server during the setup phase could be *common for multiple clients*. Thus, it can be distributed to clients using a broadcast communication channel or caching content delivery networks.

All our protocols allow for generating such a common encrypted database for multiple clients without

compromising the privacy of the server's database. Though the number of queries per client is restricted to $N_C^{\max}$, an adversary might register as k clients to run $k \cdot N_C^{\max}$ interactive queries, which can suffice for guessing low-entropy inputs. In order to disallow the clients guessing elements in the database, our protocols should only be used in scenarios where the entropy of the inputs is high, since otherwise the queries altogether can recover the server's set. As an example, it could be used in the scenario of the messaging service, since phone numbers have a relatively high entropy. However, in case of the malware checking scenario it can only be applied if high-entropy signatures of the applications are compared instead of identifiers such as application names, which follow certain patterns.

However, even though it allows the clients to perform as many queries as they want on the same (counting) BF, the client cannot exploit the fact that the same encrypted database is used, under the assumption that the underlying encryption scheme is secure.

Moreover, there is an advantage for the client when the same distributed database is used: the client can check if the server is cheating by using different inputs for different users. In contrast, when running multiple independent PSI instances, the server could use different inputs for different clients.

4.2 Security Against Malicious Client for NR-PSI and GC-PSI

In NR-PSI and the GC-PSI protocols the only messages sent by the client are those in the OTs. Hence, these protocols can easily be secured against a malicious client by not using OT precomputation and using OT extension with security against malicious receivers, e.g., that from [3, 38]. These protocols are only slightly less efficient than the passively secure OT extension of [2] which we use in our experiments.

5 Conclusion and Future Work

In this paper, we optimized the efficiency of private set intersection via precomputation so that it can efficiently deal with an unequal number of inputs. We transformed four existing protocols into the precomputation form, and give a systematic comparison of their performance in a PC setting and a smartphone setting. On the one hand, our results show that GC-PSI and DH-PSI provide the most efficient online phase in the PC setting and smartphone setting, respectively. On the other hand, GC-PSI provides the most efficient setup phase especially when AES-NI is supported.
Future work could improve the performance of GC-PSI on Android by making use of AES-NI or implement the extensions proposed in §4. Also our protocol implementations could be turned into real-world applications for the motivating scenarios described in §1 such as the malware detection service.

Acknowledgements

We thank the anonymous reviewers for their valuable feedback on the paper. This work has been co-funded by the German Federal Ministry of Education and Research (BMBF) and the Hessen State Ministry for Higher Education, Research and the Arts (HMWK) within CRISP, by the DFG as part of project E3 within the CRC 1119 CROSSING, by the Cloud Security Services (CloSer) project (3881/31/2016), funded by the Finnish Funding Agency for Innovation (TEKES), and 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.

References

[1] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner, "Ciphers for MPC and FHE," in *Advances in Cryptology – EUROCRYPT'15*, ser. LNCS, vol. 9056. Springer, 2015, pp. 430–454.

[2] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer and extensions for faster secure computation," in *ACM Computer and Communications Security (CCS'13)*. ACM, 2013, pp. 535–548.

[3] ——, "More efficient oblivious transfer extensions with security for malicious adversaries," in *Advances in Cryptology – EUROCRYPT'15*, ser. LNCS, vol. 9056. Springer, 2015, pp. 673–701.

[4] N. Asokan, A. Dmitrienko, M. Nagy, E. Reshetova, A. Sadeghi, T. Schneider, and S. Stelle, "CrowdShare: Secure mobile resource sharing," in *Applied Cryptography and Network Security (ACNS'13)*, ser. LNCS, vol. 7954. Springer, 2013, pp. 432–440.

[5] P. Baldi, R. Baronio, E. De Cristofaro, P. Gasti, and G. Tsudik, "Countering GATTACA: efficient and secure testing of fully-sequenced human genomes," in *ACM Computer and Communications Security (CCS'11)*. ACM, 2011, pp. 691–702.

[6] D. Beaver, "Precomputing oblivious transfer," in *Advances in Cryptology – CRYPTO'95*, ser. LNCS, vol. 963. Springer, 1995, pp. 97–109.

[7] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, "Efficient garbling from a fixed-key blockcipher," in *IEEE Symposium on Security and Privacy (S&P'13)*. IEEE, 2013, pp. 478–492.

[8] L. Bouncy Castle Inc., "Bouncy Castle crypto APIs," https://www.bouncycastle.org/, 2017, accessed: 2017-03-10.

[9] J. Boyar and R. Peralta, "A new combinational logic minimization technique with applications to cryptology," in *Symposium on Experimental Algorithms (SEA'10)*, ser. LNCS, vol. 6049. Springer, 2010, pp. 178–189.

[10] H. Carter, C. Amrutkar, I. Dacosta, and P. Traynor, "For your phone only: custom protocols for efficient secure function evaluation on mobile devices," *Security and Communication Networks*, vol. 7, no. 7, pp. 1165–1176, 2014.

[11] E. D. Cristofaro and G. Tsudik, "Practical private set intersection protocols with linear complexity," in *Financial Cryptography and Data Security (FC'10)*, ser. LNCS, vol. 6052. Springer, 2010, pp. 143–159.

[12] ——, "Experimenting with fast private set intersection," in *Trust and Trustworthy Computing (TRUST'12)*, ser. LNCS, vol. 7344. Springer, 2012, pp. 55–73.

[13] D. Demmler, T. Schneider, and M. Zohner, "Ad-hoc secure two-party computation on mobile devices using hardware tokens," in *USENIX Security Symposium'14*. USENIX, 2014, pp. 893–908.

[14] ——, "ABY - A framework for efficient mixed-protocol secure two-party computation," in *Network and Distributed System Security Symposium (NDSS'15)*. The Internet Society, 2015.

[15] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[16] I. Dinur, Y. Liu, W. Meier, and Q. Wang, "Optimized interpolation attacks on LowMC," in *Advances in Cryptology – ASIACRYPT'15*, ser. LNCS, vol. 9453. Springer, 2015, pp. 535–560.

[17] C. Dobraunig, M. Eichlseder, and F. Mendel, "Higher-order cryptanalysis of LowMC," in *Information Security and Cryptology (ICISC'15)*, ser. LNCS, vol. 9558. Springer, 2015, pp. 87–101.

[18] C. Dong, L. Chen, and Z. Wen, "When private set intersection meets big data: an efficient and scalable protocol," in *ACM Computer and Communications Security (CCS'13)*. ACM, 2013, pp. 789–800.

[19] L. Fan, P. Cao, J. M. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," in *SIGCOMM'98*. ACM, 1998, pp. 254–265.

[20] M. Fischlin, B. Pinkas, A. Sadeghi, T. Schneider, and I. Visconti, "Secure set intersection with untrusted hardware tokens," in *Topics in Cryptology – CT-RSA'11*, ser. LNCS, vol. 6558. Springer, 2011, pp. 1–16.

[21] F. S. Foundation, "The GNU multiple precision arithmetic library," https://gmplib.org, 2017, accessed: 2017-03-10.

[22] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold, "Keyword search and oblivious pseudorandom functions," in *Theory of Cryptography Conference (TCC'05)*, ser. LNCS, vol. 3378. Springer, 2005, pp. 303–324.

[23] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *Advances in Cryptology – EUROCRYPT'04*, ser. LNCS, vol. 3027. Springer, 2004, pp. 1–19.

[24] P. Gasti and K. B. Rasmussen, "Privacy-preserving user matching," in *ACM Workshop on Privacy in the Electronic Society (WPES'15)*. ACM, 2015, pp. 111–120.

[25] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *ACM Symposium on Theory of Computing (STOC'09)*. ACM, 2009, pp. 169–178.

[26] N. Gilboa and Y. Ishai, "Distributed point functions and their applications," in *Advances in Cryptology – EUROCRYPT'14*, ser. LNCS, vol. 8441. Springer, 2014, pp. 640–658.

[27] D. Giry, "BlueKrypt cryptogrphic key length recommendation," http://www.keylength.com, 2017, accessed: 2017-02-28.

[28] S. D. Gordon, J. Katz, V. Kolesnikov, F. Krell, T. Malkin, M. Raykova, and Y. Vahlis, "Secure two-party computation in sublinear (amortized) time," in *ACM Conference on Computer and Communications Security (CCS'12)*. ACM, 2012, pp. 513–524.

[29] L. Grassi, C. Rechberger, D. Rotaru, P. Scholl, and N. P. Smart, "MPC-friendly symmetric key primitives," in *ACM Computer and Communications Security (CCS'16)*. ACM, 2016, pp. 430–443.

[30] C. Hazay and Y. Lindell, "Constructions of truly practical secure protocols using standard smartcards," in *ACM Computer and Communications Security (CCS'08)*. ACM, 2008, pp. 491–500.

[31] ——, "Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries," in *Theory of Cryptography Conference (TCC'08)*, ser. LNCS, vol. 4948. Springer, 2008, pp. 155–175.

[32] W. Henecka and T. Schneider, "Faster secure two-party computation with less memory," in *Computer and Communications Security (ASIACCS'13)*. ACM, 2013, pp. 437–446.

[33] Y. Huang, P. Chapman, and D. Evans, "Privacy-preserving applications on smartphones," in *USENIX Workshop on Hot Topics in Security (HotSec'11)*. USENIX, 2011.

[34] Y. Huang, D. Evans, and J. Katz, "Private set intersection: Are garbled circuits better than custom protocols?" in *Network and Distributed System Security Symposium (NDSS'12)*. The Internet Society, 2012.

[35] B. A. Huberman, M. K. Franklin, and T. Hogg, "Enhancing privacy and trust in electronic communities," in *ACM Conference on Electronic Commerce (EC'99)*, 1999, pp. 78–86.

[36] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *Advances in Cryptology – CRYPTO'03*, ser. LNCS, vol. 2729. Springer, 2003, pp. 145–161.

[37] S. Jarecki and X. Liu, "Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection," in *Theory of Cryptography Conference (TCC'09)*, ser. LNCS, vol. 5444. Springer, 2009, pp. 577–594.

[38] M. Keller, E. Orsini, and P. Scholl, "Actively secure OT extension with optimal overhead," in *Advances in Cryptology – CRYPTO'15*, ser. LNCS, vol. 9215. Springer, 2015, pp. 724–741.

[39] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, "Efficient batched oblivious PRF with applications to private set intersection," in *ACM Computer and Communications Security (CCS'16)*. ACM, 2016, pp. 818–829.
[40] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," in *International Colloquium on Automata, Languages and Programming (ICALP'08)*, ser. LNCS, vol. 5126. Springer, 2008, pp. 486–498.
[41] E. Kushilevitz and R. Ostrovsky, "Replication is NOT needed: SINGLE database, computationally-private information retrieval," in *Foundations of Computer Science (FOCS '97)*. IEEE Computer Society, 1997, pp. 364–373.
[42] Y. Lindell and B. Pinkas, "A proof of security of Yao's protocol for two-party computation," *Journal of Cryptology*, vol. 22, no. 2, pp. 161–188, 2009.
[43] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi, "ObliVM: A programming framework for secure computation," in *Symposium on Security and Privacy (S&P'15)*. IEEE Computer Society, 2015, pp. 359–376, implementation available at: https://github.com/oblivm/ObliVMGC.
[44] C. A. Meadows, "A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party," in *IEEE Symposium on Security and Privacy (S&P'86)*. IEEE, 1986, pp. 134–137.
[45] T. Meskanen, J. Liu, S. Ramezanian, and V. Niemi, "Private membership test for Bloom filters," in *International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom'15)*. IEEE, 2015, pp. 515–522.
[46] S. Nagaraja, P. Mittal, C. Hong, M. Caesar, and N. Borisov, "Botgrep: Finding P2P bots with structured graph analysis," in *USENIX Security Symposium'10*. USENIX, 2010, pp. 95–110.
[47] M. Nagy, E. D. Cristofaro, A. Dmitrienko, N. Asokan, and A. Sadeghi, "Do I know you?: efficient and privacy-preserving common friend-finder protocols and applications," in *Annual Computer Security Applications Conference (ACSAC'13)*, 2013, pp. 159–168.
[48] M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions," *J. ACM*, vol. 51, no. 2, pp. 231–262, 2004.
[49] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh, "Location privacy via private proximity testing," in *Network and Distributed System Security Symposium (NDSS'11)*. The Internet Society, 2011.
[50] R. Nojima and Y. Kadobayashi, "Cryptographically secure Bloom-filters," *Trans. Data Privacy*, vol. 2, no. 2, pp. 131–139, 2009.
[51] A. Pagh, R. Pagh, and S. S. Rao, "An optimal Bloom filter replacement," in *ACM-SIAM Symposium on Discrete Algorithms (SODA'05)*. SIAM, 2005, pp. 823–829.
[52] A. Partow, "Bloom filter implementation," https://github.com/ArashPartow/bloom, 2017, accessed: 2017-03-10.
[53] B. Pinkas, T. Schneider, G. Segev, and M. Zohner, "Phasing: Private set intersection using permutation-based hashing," in *USENIX Security Symposium'15*. USENIX, 2015, pp. 515–530.
[54] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams, "Secure two-party computation is practical," in *Advances in Cryptology – ASIACRYPT'09*, ser. LNCS, vol. 5912. Springer, 2009, pp. 250–267.
[55] B. Pinkas, T. Schneider, and M. Zohner, "Faster private set intersection based on OT extension," in *USENIX Security Symposium'14*. USENIX, 2014, pp. 797–812.
[56] ——, "Scalable private set intersection based on OT extension," *IACR Cryptology ePrint Archive*, vol. 2016/930, 2016, http://ia.cr/2016/930.
[57] S. Ramezanian, "A Study of Privacy Preserving Queries with Bloom Filters," Master's thesis, University of Turku, Finland, 2016.
[58] K. Shimizu, K. Nuida, H. Arai, S. Mitsunari, N. Attrapadung, M. Hamada, K. Tsuda, T. Hirokawa, J. Sakuma, G. Hanaoka, and K. Asai, "Privacy-preserving search for chemical compound databases," *BMC Bioinformatics*, vol. 16, no. 18, p. S6, 2015.
[59] R. Sion and B. Carbunar, "On the practicality of private information retrieval," in *Network and Distributed System Security Symposium (NDSS'07)*. The Internet Society, 2007.
[60] S. Tamrakar, J. Liu, A. Paverd, J. Ekberg, B. Pinkas, and N. Asokan, "The circle game: Scalable private membership test using trusted hardware," in *ACM Asia Computer and Communications Security (AsiaCCS'17)*. ACM, 2017, pp. 31–44.
[61] A. C.-C. Yao, "How to generate and exchange secrets," in *Foundations of Computer Science (FOCS'86)*. IEEE, 1986, pp. 162–167.
[62] A. C. Yao, "Protocols for secure computations (extended abstract)," in *Foundations of Computer Science (FOCS'82)*. IEEE, 1982, pp. 160–164.
[63] S. Zahur, M. Rosulek, and D. Evans, "Two halves make a whole - reducing data transfer in garbled circuits using half gates," in *Advances in Cryptology – EUROCRYPT'15*, ser. LNCS, vol. 9057. Springer, 2015, pp. 220–250.

A Background

In this section, we review the necessary background information for the protocols described in §2.

A.1 Oblivious Transfer

1-out-of-2 *oblivious transfer* (OT) enables a message exchange between two parties in an oblivious way: the *sender* inputs two messages s_0 and s_1 while the *receiver* inputs a choice bit b, after which the receiver receives only the message corresponding to its choice bit, s_b. This happens in an oblivious way, i.e., on the one hand, the sender does not get to know which message was received by the receiver, who on the other hand, only learns s_b but nothing about s_{1-b}.

A method to efficiently generate a large number of OTs, called OT *extension*, was proposed in [36]. The

idea is that instead of computing a large number of OTs using expensive public-key operations, it is possible to pre-compute only a small number of so-called *base OTs*, from which any polynomial number of OTs can be computed using only efficient symmetric-key operations. OT extension with security against semi-honest adversaries was introduced in [36] and its state-of-the-art improvements are described in [2]. OT extension secure against malicious adversaries is only slightly more expensive, cf. [3, 38].

Oblivious transfers can also be precomputed efficiently, by shifting the computationally expensive operations into the offline phase as described in [6]. In the offline phase, the OT protocol is run on randomly chosen bits and messages, which are used in the online phase to mask the actual inputs. The outputs of the OTs are calculated using only simple XOR operations.

More specifically, a single OT can be precomputed as follows [6]: in the offline phase, the parties run an OT on random messages m_0 and m_1 provided by the sender, and a random bit r provided by the receiver, such that the receiver obtains m_r. Then, in the online phase, the receiver sends his choice bit b in a blinded form as $b' = r \oplus b$. The sender, given its true messages s_0 and s_1 computes the following blinded values depending on the value of b': If b' is 0, the sender computes $s'_0 = s_0 \oplus m_0$ and $s'_1 = s_1 \oplus m_1$. In the other case, when the received b' is 1, the sender computes $s'_0 = s_1 \oplus m_0$ and $s'_1 = s_0 \oplus m_1$. The sender sends the computed s'_0 and s'_1 values to the receiver, who then retrieves s_b by unblinding the message $s_b = s'_b \oplus m_r$.

A.2 Yao's Garbled Circuit

Yao's *garbled circuit* (GC) protocol was proposed in [61, 62] and is a generic secure two-party computation protocol for evaluating any function represented as a Boolean circuit. In this protocol, the *garbler* garbles the circuit by assigning symmetric keys to the input wires and encrypting the output wires with the keys on the input wires. We denote the GC generation process by $(\widetilde{C_x}, s_{i,0}, s_{i,1}) \leftarrow GC.Build(C, x)$, where C denotes the circuit, x the garbler's input, $\widetilde{C_x}$ the garbled circuit and $s_{i,0}$ and $s_{i,1}$ ($i \in \{1, \ldots, |y|\}$) denote the key pairs for the *evaluator*'s input y.

The garbler stores $s_{i,0}$ and $s_{i,1}$ ($i \in \{1, \ldots, |y|\}$) and sends over the garbled circuit $\widetilde{C_x}$ to the evaluator. Then, the garbler and the evaluator run an OT protocol, throughout which the evaluator obliviously asks for the keys corresponding to its input bits, i.e., the evaluator receives $s_{i,y[i]}$ for every $i \in \{1, \ldots, |y|\}$. Using this data, the evaluator can compute the garbled circuit by decrypting the wires one by one. We denote the evaluation by $C(x, y) \leftarrow GC.Eval(\widetilde{C_x}, s_{i,y[i]}\ \forall i \in \{1, \ldots, |y|\})$, where $C(x, y)$ is the evaluation result.

We use Yao's garbled circuit protocol [61] with state-of-the-art optimizations: free XOR [40], fixed-key AES garbling [7], and half-gates [63]. The free-XOR optimization allows for the evaluation of linear (XOR) gates without communication or cryptographic operations and therefore the complexity only depends on the number of non-linear (AND) gates. Using fixed-key AES permutation for garbling allows for AES encryptions without having to perform the key schedule multiple times. The half-gates technique reduces the communication 2 ciphertexts per AND gate.

A.3 Bloom Filter

A *Bloom filter* consists of an n-bit array initialized with zeros, together with l independent hash functions whose output is uniformly distributed over $[0, n-1]$. To insert an element x ($BF.Insert(x)$), we compute l hash values, and set the corresponding positions to 1. To check if an element is in the Bloom filter ($\{0,1\} \leftarrow BF.Check(x)$), l positions are calculated in the same way. If any of these positions is 0, we can conclude that the element has not been added. Otherwise, the element is declared to be added. For a false positive rate (FPR) of $2^{-\epsilon}$, an optimized Bloom filter needs $1.44\epsilon N$ bits to store N elements [51]. Note that this is better (for a large N) than a naive storage of N hash values which would require at least $\Omega(N \log N)$ bits.

In some scenarios like malware checking, it is acceptable to exhibit a small false-positive rate. A user who receives a positive result will reveal the intersection to ascertain the result and request further information. If the FPR is sufficiently low, a small fraction of apps is not enough for the dictionary provider to profile users.

B Asymptotic Efficiency

Tab. 6 details the communication between $\mathcal{S}$ and $\mathcal{C}$ in the different phases and gives the (minimum) storage capacity needed by $\mathcal{C}$.

	Base Phase	Setup Phase	Online Phase	Client Storage
RSA-PSI (Fig. 1)	-	$1.44\epsilon N_S$	$2mN_C$	$1.44\epsilon N_S + 2m(1+N_C^{\max})$
DH-PSI (Fig. 2)	-	mN_S	$2mN_C$	$m + 1.44\epsilon N_S + mN_C$
NR-PSI (Fig. 3)	$\sigma(5m+2n) + \sigma n + 2n^2 N_C^{\max} + nN_C^{\max}(2n+1) + nN_C^{\max}$	$1.44\epsilon N_S$	$nN_C(1+2m)$	$nN_C^{\max}(1+n) + 1.44\epsilon N_S$
GC-PSI (Fig. 4)	$\sigma(5m+2n) + \sigma n + 2n^2 N_C^{\max} + nN_C^{\max}(2n+1) + N_C^{\max}(n+5{,}440 \cdot 2 \cdot 128)$	$1.44\epsilon N_S$	$nN_C(1+2m)$	$N_C^{\max}(n^2 + 5{,}440 \cdot 2 \cdot 128) + 1.44\epsilon N_S$

Table 6. Communication in the different protocols and the minimal storage capacity required by $\mathcal{C}$, where σ denotes the number of base OTs in OT extension, n the message length, m the length of the ciphertexts, ϵ the FPR parameter for the Bloom filter, $N_C^{\max}$ the upper bound of the client's number of inputs, and N_C and N_S the client's and the server's number of inputs, respectively. For GC-PSI, the AES garbled circuit has 5,440 AND gates.

C Further Extensions

In this section, we describe further extensions besides the two described in §4. Specifically, we describe how we can store the server's database in a secret-shared form (cf. §C.1), how we can generate the garbled circuits in a distributed manner (cf. §C.2), or allow for private keyword search on encrypted data (cf. §C.3).

C.1 Distributed Data Encryption for GC-PSI

In some scenarios, $\mathcal{S}$'s database may be highly sensitive and should be protected against malicious insiders. For example, in the cloud-based malware checking scenario, the anti-malware vendor may put its malware database on a lookup server that is operated by a third party, such as a content delivery network. Since the database is the main intellectual property of the anti-malware vendor, its content should be hidden from the lookup server.

To solve this problem, we introduce two semi-honest servers $\mathcal{S}_1$ and $\mathcal{S}_2$ that hold shares of the original database $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_{N_S}\}$, such that $\mathbf{x}_i = \mathbf{x}_{i,1} \oplus \mathbf{x}_{i,2}$, where $\mathcal{S}_1$ holds $\{\mathbf{x}_{1,1}, ..., \mathbf{x}_{N_S,1}\}$ and $\mathcal{S}_2$ holds $\{\mathbf{x}_{1,2}, ..., \mathbf{x}_{N_S,2}\}$. In addition, each server $\mathcal{S}_i$ holds a secret key k_i. We assume that at most one of these two servers can be passively corrupted.

Then, the two servers $\mathcal{S}_1$ and $\mathcal{S}_2$ jointly encrypt the database by computing $AES_{k_1 \oplus k_2}(\mathbf{x}_{i,1} \oplus \mathbf{x}_{i,2})$. Finally, one of the servers (say $\mathcal{S}_1$) inserts the encrypted values in a Bloom filter and sends it to the client $\mathcal{C}$ (as in the setup phase of the original protocol in Fig. 4).

According to [1, Table 7], the ABY framework [14] can securely and in parallel evaluate 100,000 blocks of AES with the GMW protocol in 556 seconds on two standard PCs. Hence, jointly encrypting a DB with 2^{20} elements would take about 1.6 hours in the setup phase, which is a one-time expense.

In the online phase, $\mathcal{C}$ secret shares the element $\mathbf{y}$ into two shares such that $\mathbf{y} = \mathbf{y}_1 \oplus \mathbf{y}_2$ and sends $\mathbf{y}_i$ to $\mathcal{S}_i$. Then, $\mathcal{S}_1$ and $\mathcal{S}_2$ jointly compute $AES_{k_1 \oplus k_2}(\mathbf{y}_1 \oplus \mathbf{y}_2)$ in the same way as it was in the setup phase. However, this time, $\mathcal{S}_2$ obtains the result and sends it to $\mathcal{C}$, who checks if it is in the Bloom filter.

In this solution $\mathcal{C}$ has negligible workload in terms of both computation and communication as all cryptographic operations are outsourced to $\mathcal{S}_1$ and $\mathcal{S}_2$. However, $\mathcal{C}$ has to trust the service provider that nobody has access to both $\mathcal{S}_1$ and $\mathcal{S}_2$ simultaneously, since in that case one can recover $\mathcal{C}$'s input $\mathbf{y} = \mathbf{y}_1 \oplus \mathbf{y}_2$.

C.2 Distributed Garbled Circuit Generation for GC-PSI

Instead of having $\mathcal{C}$ split $\mathbf{y}$ into shares and send each share to each server, we can let the two data servers $\mathcal{S}_1$ and $\mathcal{S}_2$ jointly generate the garbled circuit for AES.

$\mathcal{S}_i$ chooses random wire labels $\widetilde{in}_j^i$ for the inputs in^i of the garbled circuit. Then $\mathcal{S}_1$ and $\mathcal{S}_2$ jointly generate a garbled circuit GC for evaluating $AES_{k_1 \oplus k_2}(\cdot)$ using the input wire labels $\widetilde{in}^i = \widetilde{in}_1^i \oplus \widetilde{in}_2^i$, and send it to $\mathcal{C}$. In the online phase, $\mathcal{C}$ runs 128 OT extensions with $\mathcal{S}_1$ and 128 OT extensions with $\mathcal{S}_2$ to obliviously obtain the wire labels $\widetilde{\mathbf{y}}_1$ of $\widetilde{in}_1^i$ and $\widetilde{\mathbf{y}}_2$ of $\widetilde{in}_2^i$, respectively, that correspond to his input $\mathbf{y}$. Now the client sets the input wire label $\tilde{\mathbf{y}} = \tilde{\mathbf{y}}_1 \oplus \tilde{\mathbf{y}}_2$ and evaluates the garbled circuit to obtain $AES_k(\mathbf{y})$ and checks if it is in the Bloom filter.

Compared to the online phase of GC-PSI, $\mathcal{C}$ only needs to run twice the number of OT extensions; the size of the garbled circuit is the same. The servers need to

jointly generate a GC for AES which has at least 5,120 AND gates. When using fixed-key AES garbling, half-gates, and free XOR, the cost for garbling an AND gate is dominated by 4 secure evaluations of fixed-key AES. Therefore, the two servers perform $5{,}120 \cdot 4 = 20{,}480$ secure AES evaluations to garble an AES circuit. Using the performance results from ABY [1, Table 6] for single blocks of AES, this takes about $50\ \text{ms} \cdot 20{,}480 = 17$ minutes, which is run in the base phase.

In terms of security, even when both $\mathcal{S}_1$ and $\mathcal{S}_2$ are (passively) corrupted they still do not learn any information about $\mathcal{C}$'s inputs due to the security of the two OT extension protocols. Hence, colluding $\mathcal{S}_1$ and $\mathcal{S}_2$ can only learn the database, but not the clients' queries.

C.3 Private Keyword Search on Encrypted Data for GC-PSI

Offloading the Bloom filter to the setup phase prevents the dataset from being updated frequently, which is a critical requirement for certain scenarios. In this case, $\mathcal{C}$ can run a private keyword search protocol, e.g., [26, §4.2].

Specifically, $\mathcal{S}_1$ and $\mathcal{S}_2$ jointly encrypt the database as we described in §C.1. This time both servers keep a copy of the encrypted database without sending it to $\mathcal{C}$. When $\mathcal{C}$ wants to query a value $\mathbf{y}$, it first runs the procedure in §C.2 to get $\mathbf{y} = AES_k(\mathbf{y})$. Then, $\mathcal{C}$ runs the private keyword search protocol in [26, §4.2] on $\mathbf{y}$ with both $\mathcal{S}_1$ and $\mathcal{S}_2$. If it outputs $\perp$, $\mathcal{C}$ knows that $\mathbf{y}$ is not in the database.

The disadvantage of this protocol is that the servers $\mathcal{S}_1$ and $\mathcal{S}_2$ need to do $\mathcal{O}(N_S)$ computation per query. However, any updates to the encrypted database need to be performed only on the server side and the databases of the clients do not need to be updated.

The private keyword search protocol in [26, §4.2] requires $\mathcal{S}_1$ and $\mathcal{S}_2$ to not collude with each other. Otherwise, clients' queries will be recovered.