Outsourced Private Set Intersection Using Homomorphic Encryption

Florian Kerschbaum
SAP Research
Karlsruhe, Germany
florian.kerschbaum@sap.com

ABSTRACT

Private set intersection enables two parties – a client and a server – to compute the intersection of their respective sets without disclosing anything else. It is a fundamental operation – equivalent to a secure, distributed database join – and has many applications particularly in privacy-preserving law enforcement. In this paper we present a novel protocol that has linear complexity, is secure in the malicious model without random oracles, is client set size-independent and efficient. Furthermore, the computation of the intersection can be outsourced to an oblivious service provider, as in secure cloud computing. We leverage a completely novel construction for computing the intersection using Bloom filter and homomorphic encryption. For outsourcing we require and introduce a new homomorphic encryption scheme which may be of independent interest.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Cryptographic controls*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications*

General Terms

Algorithms, Security

Keywords

Private Set Intersection, Security, Privacy, Database Join, Bloom Filter

1. INTRODUCTION

Private set intersection (PSI) enables two parties – a client and a server – to compute the intersection of their respective sets without disclosing anything about their inputs. The client will learn the intersection of the two sets and the server will learn nothing. PSI is a fundamental building block for many applications – equivalent to a secure, distributed database join. Even as a stand-alone application, it already has many real-world applications.

Imagine a federal authority maintaining a list of suspects. It may want to search several commercial databases for appearances of suspects. A typical example is the terror watch list (or no-fly list) maintained by the department of homeland security (DHS) checked against flight passengers to and from the United States. Clearly, the DHS wants to maintain confidentiality for this list, but also the innocent flight passengers may have a right to or at least an interest in privacy. Other examples include criminal investigations where the police (e.g. the FBI) wants to search databases of the DMV, IRS, or employers. PSI can also be used to realize private database queries [11]. Using PSI they could obtain the intersection of suspects and individuals in the database. The privacy of all individuals not in the set of suspects will be maintained.

Even the size of the client's set may be sensitive which is also hidden by our protocols and there may be a need to authenticate the set of suspects, e.g. by a judge of a federal court. In certified or authenticated PSI the client first authorizes its set with a trusted third party. Such certification prevents the client from obtaining the entire server's set by including all possible elements in its input.

We abstractly view PSI as the following computation. There is a client C which has a set $\{c_1, \ldots, c_v\}$ of size v and a server S which has a set $\{s_1, \ldots, s_w\}$ of size w. After the computation the client has obtained the intersection $\{c_1, \ldots, c_v\} \cap \{s_1, \ldots, s_w\}$ and the server has learnt nothing.

We optionally extend this notion of PSI to outsourced PSI ($OPSI$), as in cloud computing. There is a service provider P and both client and server submit their sets to the provider which then computes the intersection. Nevertheless, we maintain privacy, i.e. the provider does not learn anything about the inputs or the intersection. We envision a scenario where the server stores its set at the service provider, e.g. a list of convicted criminal offenders. The client may then query the database while the server is off-line. Our protocols readily extend to this scenario.

1.1 Contributions

In this paper we present several PSI protocol variants. First, we present a preliminary PSI variant secure in the semi-honest model. This variant explains our novel construction based on Bloom filter [3] and homomorphic encryption [19]. It is established practice in the database community to use Bloom filter for improved performance of distributed joins in databases [26, 30]. Bloom filter allow for

ASIACCS '12, May 2–4, 2012, Seoul, Korea.

false positives when testing set inclusion, but the probability of a false positive can be made arbitrarily small using a parameter k. Our scheme has a second source of false positives (the SYY technique in [32]), but this has also an (independent) parameter l.

We then present our main result: a variant secure in the malicious model ($MPSI$). The $MPSI$ variant additional requires certification of the client set. Finally, we present the $OPSI$ variant for outsourcing the computation of the intersection to an oblivious service provider P. The service provider may not learn anything about the inputs or the output (the intersection), but should perform most of the computation. This is a common setting in cloud computing where service providers offer computational resources. In order to enable this outsourcing we introduce a novel construction for homomorphic encryption which may be of independent interest.

The protocol variants we propose have a number of advantages compared to the state of the art. Particularly, we combine the security and performance advantages of almost all previous protocols in one protocol. Our protocols are

1. *secure*, because
   (a) the $MPSI$ variant is provably *secure in the malicious model*. The $OPSI$ variant is secure in the semi-honest model, since security in the malicious model would undo all advantages of outsourcing the computation. Previous PSI protocols also secure in the malicious model are [9, 10, 15, 20, 21, 22, 25].
   (b) they require only *minimal trust assumptions*. We are secure in the standard model (without random oracles). Previous PSI protocols also secure in the standard model are [9, 21, 22, 25]. Our only cryptographic assumptions is the quadratic residuosity assumption. This is one of the weakest assumptions one can make and to the best of our knowledge our protocols are the first PSI protocols to only make this assumption.
   (c) they are *client set size-independent*. Although we reveal an upper bound on the size of the client set that is different from the security parameter, the computational effort of the server is independent of this bound. This is different than client set size-hiding from the previous work of [1] where the upper bound of the client set coincides with the security parameter and is therefore indiscernible.
   (d) the $MPSI$ variant uses *certified client sets*. The client has to certify its (entire) set with a trusted third party who verifies that it adheres to some policy. This prevents the client from "stealing" the server's input by submitting a set with all possible elements as its input. This was first introduced in [5] and later used as authenticated client sets in [12] where certification is performed on single elements only. The certification of the entire set has the advantage that the client cannot omit single elements and the trusted third party can therefore enforce stricter policies.
2. *efficient*, because
   (a) the PSI and $MPSI$ variants have linear complexity ($O(k(v+w))$). Previous protocols with linear $O(v+w)$ complexity are [12] (secure in the semi-honest model) and [10, 20, 22] (secure in the malicious model). The $OPSI$ variant has quadratic complexity ($O(kw^2)$). Both complexities are optimal, if in the $OPSI$ variant there is only one key (used for the homomorphic encryption scheme) and the provider is not supposed to learn any information including the intersection's size.
   (b) the PSI and $MPSI$ variants do not use expensive operations. We do not use modular exponentiations or full-domain hash functions (which also require modular exponentiations [8]). Instead we only use modular multiplications. The $OPSI$ variant uses bilinear maps on elliptic curves, but the computation is performed by a powerful service provider.
   (c) the $MPSI$ variant uses a similar (absolute) number of normalized modular multiplications as the most efficient, linear-complexity protocols. De Cristofaro et al. [10] perform an extensive analysis and De Cristofaro and Tsudik [12] present implementation results. We use a similar number of modular multiplications as the best protocol in [10]. Nevertheless, as stated above we do not use full-domain hashing (which has been ignored in the analysis in [10]) and require much less precomputation. In [10] De Cristofaro et al. assume pre-computation of fixed-base modular exponentiations which may become a problem when dealing with multiple protocol instances with different parameters.
3. are the first that can be *outsourced* to an oblivious service provider where
   (a) both client and server only submit encrypted input and the service provider performs the computation *obliviously* without learning anything about the input (including the client set size) and the intersection (including its size).
   (b) the computation can be performed *independently*. Independent from either the client's or server's on-line availability, i.e. either client or server store their (encrypted) input at the service provider and the other party can query with their input any time when required.

The remainder of the paper is structured as follows: We next present related work on private set intersection in Section 2. In Section 3 we review the homomorphic encryption schemes used. We describe the protocols in Section 4 and extend them to outsourcing in Section 5. We present our conclusions in Section 6.

2. RELATED WORK

2.1 Security Models

Before reviewing previous *PSI* protocols we briefly explain the security models. Security in *PSI* is evaluated by comparison to an ideal model. In the ideal model, both client and server submit their input to a trusted third party which performs the intersection and returns the result (to the client). This is compared to the real model implementing the protocol.

According to the definitions by Goldreich [18] a *semi-honest* adversary in the real model adheres to the protocol by faithfully providing correct input, but may keep a record of all interactions and later try to infer additional information from it. Security against a semi-honest adversary requires that client and server in the real model do not learn more information than in the ideal model, if they behave according to the protocol specification.

To the contrary, a *malicious* adversary may behave arbitrarily in the real model. Security against a malicious adversary implies that client and server in the real model always do not learn more information than in the ideal model. For every adversary in the real model there must be an adversary in the ideal model that simulates the same behavior. Nevertheless, malicious adversaries may still refuse to participate, abort prematurely or (worse) modify their input. We offer client set certification to partially address this problem.

2.2 Private Set Intersection

PSI can be realized using general secure computation [33], but specialized protocols are more efficient. Several such specialized protocols have been proposed in the literature. We group them by their basic technique.

PSI using Oblivious Polynomial Evaluation.

Using oblivious polynomial evaluation [27] one can evaluate a polynomial without disclosing the coefficients. The idea is to represent the set elements as the roots of the polynomial and then obliviously evaluate it on the other party's set elements. Freedman et al. [15] present protocols secure against semi-honest and malicious adversaries (in the random oracle model). The computation complexity is quadratic $O(vw)$, although the number modular exponentiations can be reduced to $O(w \log\log v)$. Kissner and Song [25] extend the construction to additional set operations and multiple players. They present protocols secure against semi-honest and malicious adversaries (in the standard model) using generic zero-knowledge proofs. The computation complexity remains quadratic at $O(vw)$. Dachman-Soled et al. [9] presented an improved construction secure against malicious adversaries based on [25] without generic zero-knowledge proofs, although this increases the complexity. Hazay and Nissim [21] also present improved zero-knowledge proofs. Their construction is secure against malicious adversaries in the standard model and has $O(v + w(\log\log v + \log w))$ complexity, which can be further reduced in the random oracle model.

While some of these protocols already achieve security against malicious adversaries in the standard model, none of them achieves linear complexity as our protocols do.

PSI using Oblivious Pseudo-Random Functions.

Using oblivious pseudo-random functions [14] the client can evaluate a keyed, pseudo-random function on its input where the server holds the key. The idea is to compute the intersection on the pseudo-random functions of the set elements. The client obtains the result of the pseudo-random function obliviously. Hazay and Lindell [20] present the first such protocols secure against semi-honest and malicious adversaries. These are the first linear $O(v + w)$ complexity protocols. Jarecki and Liu [22, 23] further improved these protocols.

While these protocols already achieve security in the standard model and linear complexity, their constants are quite high and have been improved subsequently. We furthermore provide client set certification, client set size-independence and outsourcing.

PSI using Blind Signatures.

Using blind signatures [6] a client can obtain a signature on its input without disclosing it. The idea is to present (aggregate) signatures of the elements of a set, hash the result of the verification and compute the intersection on the hashes. De Cristofaro and Tsudik [12] present protocols secure against semi-honest adversaries. These protocols also have linear $O(v+w)$ complexity. De Cristofaro et al. [10] extend the protocols to security against malicious adversaries in the random oracle model. They maintain linear complexity and show that they have significantly lower constants than [22].

Our protocols also have linear complexity and similar constants, but do not need full domain hash functions and are secure in the standard model. De Cristofaro et al. [10] ignored the costs of full-domain hash functions in their analysis.

De Cristofaro et al. [10] also use the notion of authenticated *PSI* where the client has to present its input to a trusted third party. This has first been introduced for mutual authentication (certification) by Camenisch and Zaverucha [5], but their protocol has quadratic $O(vw)$ complexity. Ateniese et al. [1] later also present a protocol loosely based on [12] which is client set size-hiding. It has complexity $O(w + v \log v)$. They also present a variant in which the client complexity can be reduced to $O(v)$, but incurs a penalty polynomial in the security parameter.

Furthermore, we are not aware of previous protocols where the computation can be outsourced to an oblivious service provider.

2.3 Bloom Filters in Cryptography

We present a novel construction for *PSI* using Bloom filter [3]. Bloom filter have been used in cryptography before.

Bellovin and Cheswick [2] and independently Goh [17] use Bloom filter to securely search documents. It enables checking whether a document contains certain keywords without disclosing any of them. Their protection mechanism is to compute the hash function as a cryptographic pseudo-random function. This technique has been used in [31] to securely query a database and in [29] for privacy-preserving data mining.

Nojima and Kadobayashi [28] present an interactive protocol for securely checking set inclusion via Bloom filter without disclosing the Bloom filter or the checked element. They use blind signatures or oblivious pseudo-random functions

in order to compute the hash functions of the Bloom filter. Their protocols are only secure against semi-honest adversaries. While these also have linear $O(w)$ complexity, they increase the constants compared to PSI based on blind signatures by the number of hash functions. We present a completely novel construction based on homomorphic encryption which is secure against malicious adversaries in the standard model and has significantly reduced constants.

Kerschbaum [24] recently presented techniques to check public-key encrypted Bloom filter using zero-knowledge proofs. They enable to non-interactively check for the inclusion or exclusion of an element from a Bloom filter without disclosing the Bloom filter content. They similarly use homomorphic encryption for protecting the content of the Bloom filter. These protocols are complementary to the ones presented in this paper, since, although they are non-interactive, they require to disclose the checked elements, i.e. one party's input. In this paper we present secure protocols which provide protect both parties' input and furthermore can be outsourced without requiring trust in the service provider.

3. CRYPTOGRAPHIC BUILDING BLOCKS

3.1 Goldwasser Micali Encryption

Goldwasser-Micali (GM) encryption [19] is a public-key, semantically-secure (IND-CPA) and homomorphic encryption scheme. GM encryption uses quadratic residuosity modulo a composite of two large primes p and q (RSA modulus) to encrypt one bit of plaintext. A quadratic residue r is a number, such that there exists a number s: $s^2 = r \bmod n$. GM encodes a 1 as a quadratic non-residue and a 0 as a quadratic residue. Particularly, the quadratic non-residues are pseudo quadratic residues, i.e. their Jacobi symbols are all 1. The hardness of differentiating pseudo quadratic residues from quadratic residues is known as the quadratic residuosity assumption.

We can summarize the operations as follows

KeyGen(κ): Let κ be a security parameter. Given κ generate the private key $sk = \{p, q\}$ and the public key $pk = \{n = pq, u\}$ where u is a pseudo quadratic residue.

Encrypt(x, pk): Given plaintext x and public key pk produces ciphertext c. To encrypt a 0 choose a random number r and compute $r^2 \bmod n$ (a quadratic residue). To encrypt a 1 also choose a random number r and computes $ur^2 \bmod n$ (a quadratic non-residue).

Decrypt(c, sk): Given ciphertext c and private key sk produces plaintext x. Compute the Legendre symbol $L\left(\frac{x}{p}\right)$ and decide $x = 1$, if $L\left(\frac{x}{p}\right) = -1$ and $x = 0$, if $L\left(\frac{x}{p}\right) = 1$.

Let $E(x)$ denote encryption of x under GM public key pk and let $D(c)$ denote the corresponding decryption. Multiplying two ciphertexts, e.g. $E(x) \cdot E(y)$, results in an encryption of the exclusive-or denoted by $\oplus$.

$$D(E(x) \cdot E(y)) = x \oplus y$$

GM encryption is semantically-secure (IND-CPA), i.e. one cannot infer from a ciphertext and the public key whether the ciphertext has a specific plaintext, e.g. by encrypting the plaintext and then comparing it.

3.2 Sander Young Yung Technique

Sander, Young and Yung (SYY) [32] introduce a technique that allows the computation of one logical-and operation on ciphertexts. The input ciphertexts are encrypted using GM encryption. Recall that we can perform any number of exclusive-or operations on the ciphertexts. A ciphertext $E(x)$ is first expanded as follows:

Expand(c, pk): Given GM ciphertext $c = E(x)$ and public key pk computes an expanded ciphertext $c^l = E^l(x)$. We compute $E(e_i)$ repeatedly l times $(0 \leq i < l)$.

1. Flip a fresh random coin $r_i \in \{0, 1\}$ $(i = 1, \ldots, l)$.

2. Choose a ciphertext corresponding to the plaintext e_i according to the random coin and set

$$E(e_i) = \begin{cases} E(x) \cdot E(1) = E(x \oplus 1) & \text{if } r_i = 0 \\ E(0) & \text{if } r_i = 1 \end{cases}$$

The result is an l-length vector $E^l(x) = E(e_1), \ldots, E(e_l)$. If the input ciphertext is $x = 1$, then $x \oplus 1 = 0$ and $E^l(x)$ is an all 0s vector, i.e. $e_i = 0$ $(0 \leq i < l)$. Otherwise, if the input ciphertext $x = 0$, then e_i is uniformly distributed in $\{0, 1\}$.

Decrypt(c^l, sk): In order to decrypt $E^l(x)$, one decrypts each $E(e_i)$ and if all $e_i = 0$, decides $x = 1$ and if any $e_i = 1$, decides $x = 0$. Note that there is a small probability 2^{-l} to falsely decrypt a ciphertext $E^l(0)$ as a 1. We denote this decryption operation as $D^l(c^l)$.

One can now compute a logical-and of any number of expanded ciphertexts, e.g. $E^l(x) = E(e_1), \ldots, E(e_l)$ and $E^l(y) = E(f_1), \ldots, E(f_l)$. We compute the pair-wise product of the ciphertexts, i.e.

$$\begin{aligned} E^l(x \wedge y) &= E^l(x) \times E^l(y) \\ &= E(e_1) \cdot E(f_1), \ldots, E(e_l) \cdot E(f_l) \\ &= E(e_1 \oplus f_1), \ldots, E(e_l \oplus f_l) \end{aligned}$$

If at least one of $E^l(x)$ or $E^l(y)$ consists of randomly distributed plaintexts in $\{0, 1\}$, then $E^l(x \wedge y)$ consists randomly distributed plaintexts. Only if both are all 0s plaintexts, then $E^l(x \wedge y)$ has all 0s plaintexts (except with negligible probability in l).

4. PRIVATE SET INTERSECTION

4.1 Bloom Filter

Bloom filter [3] provide a space- and time-efficient mean to check the inclusion of an element in a set. An empty Bloom filter b consists of m bits, all set to 0, and k hash functions h_i $(0 \leq i < k)$. Note that we do not require the hash functions to be random oracles. We write b_j $(0 \leq j < m)$ for the j-th bit of Bloom filter b. Bloom filter support the operations $Add(x)$ for addition of element x to the set and $Test(x)$ to test for inclusion of element x.

Create(m): m bits $(0 \leq j < m)$ are set to 0

$$\forall j.b_j = 0$$

and k hash functions h_i $(0 \leq i < k)$ are published

$$\forall i.h_i : \{0, 1\}^* \mapsto \{0, \ldots, m - 1\}$$

Add(x): The element x is hashed with all k hash functions h_i and the k bits at the resulting indices g_i are set to 1.

$$\forall i.g_i = h_i(x) \Longrightarrow b_{g_i} = 1$$

Test(x): Again, the element x is hashed with all k hash functions h_i and if all k bits at the resulting indices g_i are set, then the test function returns 1 (true).

$$\bigwedge_{i=0}^{k-1} b_{h_i(x)} \tag{1}$$

Bloom filter have a small probability for false positives, i.e. $Test(x)$ may return true, although x has never been added. The more elements are added to the set, the more likely false positives are. Given the number w of elements to be added and a desired maximum false positive rate 2^{-k}, one can compute the necessary size m of the Bloom filter as

$$m = \frac{wk}{\ln^2 2}$$

4.2 Protocol

The basic idea of our protocol is to send a Bloom filter for the client set (bit-wise) encrypted using GM encryption. Then we evaluate the *Test* function (Equation 1) for each element in the server set using the SYY technique. Finally, we compute the exclusive-or between the expanded ciphertext result and the server's element and return it.

Let $s_{i,j}$ denote the j-th bit of the server's element s_i. In the same way as we can expand the Bloom filter using the SYY technique, we can also construct an expanded vector of each of the server's elements. Then the expanded ciphertext of an element is

$$E^l(s_i) = E(s_{i,1}), \ldots, E(s_{i,l})$$

We use the same notation as for the SYY technique in order to stress that we can perform common operations on both of these ciphertexts. We can decrypt the expanded ciphertext bit-wise and reconstruct s_i. We denote a logical-and of fan-in more than two as

$$\prod_{i=1}^{n} E^l(x_i) = E^l(x_1) \times \ldots \times E^l(x_n)$$

Figure 1 shows our private set intersection protocol (secure in the semi-honest model).

The returned elements s'_1 are either elements in the intersection (if all Bloom filter bits are 1, i.e. $\bigwedge_{j=0}^{k-1} b_{h_j(s_i)} = 1$) or randomly chosen elements in $\{0,1\}^l$. Ideally, the parameter l should be chosen larger than the logarithm of the input domain size. Then, the server's elements are padded with 0s. It still remains crucial (in the malicious model) that the client performs the set intersection and does not rely on the trailing 0s for the identification of the elements in the intersection.

4.2.1 Security Proof

We prove security by comparison between the real model and an ideal model. The real model is the execution of our PSI protocol. The ideal model – as in many other secure computations – consists of a trusted server implementing the set intersection functionality $\cap$. The trusted server receives the input from both – client $c_1, \ldots, c_v$ and server $s_1, \ldots, s_w$, respectively – and returns to the client the intersection of their sets $\{c_i\} \cap \{s_j\}$. The server obtains no output. The ideal model remains for all security proofs.

In a semi-honest protocol the clients do not deviate from the protocol, therefore following Goldreich's proof construction [18] we only need to simulate their views. A view is the messages (and coin tosses) a party receives during protocol execution. The simulator may use the party's input and output in order to create an (computationally) indistinguishable simulation of the view. This proves that the party cannot infer any additional information (except its input and output).

THEOREM 1. *If the quadratic residuosity assumption holds, then protocol PSI implements private set intersection in the semi-honest model.*

PROOF. The resulting security proof in the semi-honest model is quite simple. The server does obtain any information, since all its messages are encrypted (if the quadratic residuosity assumption holds). Its view can therefore be simulated using ciphertexts only. These are all independent due to IND-CPA security of the GM encryption scheme.

The client only receives the messages for its output, i.e. the intersection. A simulator of its view is therefore trivially its output. □

4.3 Malicious Model

The $MPSI$ variant secure in the malicious model uses client set certification. In an $MPSI$ protocol the client presents his set to a trusted third party which attests that this is the set used in the PSI protocol.

Let $S(x)$ denote the signature of x by the trusted third party. Furthermore, the trusted third party now generates the key n, u for the GM encryption. This ensures that the key is trustworthy and can be used by both client and server.

In the malicious model we need to account for the false positives of a Bloom filter. A client could present a set to the trusted third party and have it certified while knowledgeable about false positives of this set with unintended privacy leakages. The client could, for example, probe for specific values at the server. We need to prevent the client from maliciously generating hash collisions.

Hence, the trusted third party also generates an exponent e that it only shares with the server S. All elements are (RSA) encrypted using this exponent before added to the Bloom filter. Nevertheless, we do not require the RSA assumption to hold as [10] does, since the client does not know e. Figure 2 shows the key generation and distribution by the trusted third party.

In the $MPSI$ protocol the trusted third party certifies the client set and submits signatures to the client. The client submits the signature $S(E(b_1), \ldots, E(b_m))$ to the server for verification during the PSI protocol. Of course, the server also encrypts his set with the secret exponent by the trusted third party before performing the encrypted Bloom filter test operation. Figure 3 shows the interaction between client and trusted third party.

4.3.1 Security Proof

We prove the security of the $MPSI$ protocol variant. Note that while the server may not adhere to the prescribed computation, e.g. by returning an encrypted element $E^l(s_i)$ without testing the Bloom filter, it does not attack correctness of the computation. The client always locally performs the intersection with its input set. The server can only violate its own privacy by revealing additional input and therefore in most practical situations where privacy is relevant he is not even inclined to do so. Therefore there is always a server in the real model leading to the same output in

C	:	$b = Create(m)$
		$i = 1, \ldots v : b.Add(c_i)$
C $\longrightarrow$ S	:	$n, u, E(b_1), \ldots, E(b_m)$
S	:	$i = 1, \ldots, w : E^l(s'_i) = E^l(s_i) \times \prod_{j=0}^{k-1} Expand(E(b_{h_j(s_i)}))$
S $\longrightarrow$ C	:	$E^l(s'_1), \ldots, E^l(s'_w)$
C	:	$\{c_1, \ldots, c_v\} \cap \{s'_1, \ldots, s'_w\}$

Figure 1: Private Set Intersection Protocol PSI

T	:	n, u, e
T $\longrightarrow$ C	:	n, u
T $\longrightarrow$ S	:	n, u, e

Figure 2: Key Distribution in the $MPSI$ Protocol

C $\longrightarrow$ T	:	$c_1, \ldots, c_v$
T	:	$b = Create(m)$
		$i = 1, \ldots, v : b.Add({c_i}^e \bmod n)$
T $\longrightarrow$ C	:	$E(b_1), \ldots, E(b_m), S(E(b_1), \ldots, E(b_m))$
C $\longrightarrow$ S	:	$E(b_1), \ldots, E(b_m), S(E(b_1), \ldots, E(b_m))$
$\ldots$		
S	:	$i = 1, \ldots, w : E^l(s'_i) = E^l(s_i) \times \prod_{j=0}^{k-1} Expand(E(b_{h_j({s_i}^e \bmod n)}))$
$\ldots$		

Figure 3: Malicious Private Set Intersection Protocol $MPSI$

the ideal model. Loosely speaking, there is no difference between the server choosing s' as an input and the server "sneaking" s' into the result.

We again prove security by comparison between real and ideal model. The real model is the execution of the $MPSI$ protocol this time. Furthermore, client and server may now behave arbitrarily during protocol execution (except protocol abortion).

THEOREM 2. *If the quadratic residuosity assumption holds, then protocol $MPSI$ implements certified private set intersection in the malicious model.*

PROOF.
Confidentiality of the client:
All inputs are encrypted using IND-CPA secure encryption (under the quadratic residuosity assumption).
Confidentiality of the server:
The client cannot predict the encrypted element s^e for any element s of the server. It is indistinguishable from a random number without the knowledge of e. Therefore the client cannot force a hash collision – even if the hash functions of the Bloom filter are reversible – with a chosen set presented to the trusted third party. The occurrence of any collision is random.

Hence, the probability for a false positive match between the Bloom filter and an element in the server's set is

$$2^{-k} + 2^{-l}$$

The probability of falsely revealing an element by the server is therefore negligible in k or l.

To ensure security against a malicious client (server), it must be shown that for any possible client (server) behavior in the real model, there is an input that the client (server) provides to the TTP in the ideal model, such that his view in the real protocol is efficiently simulatable from his view in the ideal model.
Construction of a simulator SIM_S from a malicious real-world server $\hat{S}$:

1. The simulator SIM_S executes $KeyGen(\kappa)$ in the GM encryption.
2. The simulator SIM_S creates an all 1s Bloom filter b, i.e. $b_i = 1$ $(i = 1, \ldots, m)$. The simulator sends $n, u, E(b_1), \ldots, E(b_m)$ and simulates the signature $S(E(b_1), \ldots, E(b_m))$.
3. After receiving $E^l(s'_1), \ldots, E^l(s'_w)$ from the malicious server $\hat{S}$, the simulator SIM_S decrypts $s'_1, \ldots, s'_w$.
4. The simulator SIM_S now plays the role of the ideal server interacting with the TTP (and the ideal client). The simulator submits $s'_1, \ldots, s'_w$ to the TTP.

Since GM encryption is IND-CPA secure under the quadratic residuosity assumption, the view of the malicious server $\hat{S}$ in the simulation by SIM_S and in the real protocol are indistinguishable.
Output of (honest) real client C interacting with $\hat{S}$:
For each set $s'_1, \ldots, s'_w$ the client C receives, there is an input set $s_1, \ldots, s_w$ by the server $\hat{S}$. The client builds the intersection $\{c_1, \ldots, c_v\} \cap \{s'_1, \ldots, s'_w\}$ and each elements s'_i has been used by the simulator in the ideal model, such that the outputs are identical.
Construction of a simulator SIM_C from a malicious real-world client $\hat{C}$:

1. The simulator SIM_C now plays the role of the trusted third party T performing the certification. After receiving $c_1, \ldots, c_v$ the simulator computes the encrypted

Bloom filter. It records the set $\mathcal{C} = \{c_1, \ldots, c_v\}$ and returns the encrypted and signed Bloom filter $E(b_1)$, ..., $E(b_m)$, $S(E(b_1), \ldots, E(b_m))$.

2. The simulator SIM_C now plays the role of the real-world server. After receiving $E(b_1), \ldots, E(b_m), S(E(b_1), \ldots, E(b_m))$ the simulator verifies the signature. If it does not check, it aborts.

3. The simulator SIM_C now plays the role of the ideal client interacting with the TTP (and the ideal server). The simulator submits $\mathcal{C}$.

4. After receiving the intersection $\mathcal{I}$, the simulator SIM_C creates $w - |\mathcal{I}|$ random elements and adds them and the elements in $\mathcal{I}$ to $\mathcal{S}' = \{s'_1, \ldots, s'_w\}$. It encrypts each bit of each s'_i and sends $E(s'_{1,1}), \ldots, E(s'_{w,l})$ to the client $\hat{C}$.

Since the set $\mathcal{C}$ equals the client set $\{c_1, \ldots, c_v\}$ and the set $\mathcal{S}$ only contains the elements in $\mathcal{I}$ and randomly chosen elements, the view of the malicious client $\hat{C}$ in the simulation by SIM_C and in the real protocol are indistinguishable.

□

4.4 Data Transfer

It is often required to transfer additional data d_i (e.g. a database tuple) along with a matching element s_i. Our protocols can be easily extended to accommodate data transfer. Let $E_{K_i}(x)$ denote the symmetric encryption with key K_i. The server chooses a random key K_i in the symmetric encryption system for each element s_i. It uses the (bit-wise) expanded ciphertext $E^l(K_i)$ of the key instead of the expanded ciphertext $E^l(s_i)$ of the element in preparing its response and also returns $E_{K_i}(s_i, d_i)$ to the client. The client can now recover the key K_i, if it used to be able to recover the element s_i (i.e. in case of a match). Using the key it can decrypt the data.

4.5 Comparison

We compare our $MPSI$ protocol to the best known private set intersection protocol in the malicious model in [10]. We compare correctness, communication complexity and computational performance. Nevertheless, differently from [10] we do not require random oracles or any assumption stronger than hardness of determining quadratic residuosity.

4.5.1 Correctness

Bloom filter may produce false positives. In our case the probability of a false positive is $2^{-k} + 2^{-l}$. Note that the parameters can be chosen to make this probability arbitrarily small.

In [10] there is no correctness parameter stated, but it has an implicit one depending on its security parameter κ. In their case the probability of a false positive is $2^{-\kappa}$.

4.5.2 Communication

Depending on the correctness parameters we also increase communication complexity. We transfer $m + lw$ group elements. Assuming m is chosen depending on the client's set size, we transfer $O(kv + lw)$ group elements.

The protocol of [10] requires less communication, such that it is beneficial in congested networks. It transfers $7v+3$ group elements and $w + 1$ cryptographic hashes. Although we achieve better performance for very large client set sizes (and $k < 7ln^2 2 < 4$), these seem unreasonable assumptions in practice.

4.5.3 Performance

We now compare the performance of our $MPSI$ protocol to the performance of the best variant in [10]. A detailed analysis of the performance of all PSI protocols can be also found in [10]. They conclude that the protocol in [10] has the lowest constants of the linear complexity protocols. This is underpinned by several implementations in [12].

GM encryption only uses modular multiplication and we can prepare in a pre-computation phase all quadratic residues r^2 for randomizing the ciphertexts. The client does not need to perform any encryption – this is done during certification by the trusted third party. The server needs to expand the ciphertexts and therefore compute the negation (k modular multiplications). It does not need to randomize each ciphertext in the expanded ciphertext, but only the result (l modular multiplications). Then it needs to compute the product of the expanded ciphertexts and the element ($kl + l$ modular multiplications). The server performs this operations for each of its w elements. In summary, we have $w(kl + k + 2l)$ modular multiplications.

De Cristofaro et al. [10] achieve $240w + 960v$ modular multiplications. If we instantiate $k = 16$ and $l = 32$, then we achieve $592w$ modular multiplications.

We emphasize that the analysis in [10] ignores $v + w$ full-domain hash functions. If we – following their convention – estimate each full domain hash function as 240 modular multiplications, then their cost increases to $480w + 1200v$. Our protocol is more efficient, if the client set size v is at least one tenth of the server set size w. Furthermore, in any case the computational load on the client using our protocols is significantly smaller and therefore better suitable for computationally weak devices.

De Cristofaro et al. [10] also assume pre-computation of fixed-base modular exponentiations. This may become a problem when dealing with multiple protocol instances with different parameters. If we ignore those (and, of course, our pre-computation of quadratic residues), then our protocol is more efficient, if the client set size v is at least 6% of the server set size w.

We conclude that our protocol has similar constants to the most efficient, linear-complexity version of [10]. We remind the reader that our protocol is secure in the standard model whereas [10] requires random oracles. As the first PSI protocol we show an outsourced variant in the next section.

5. OUTSOURCING

A novel and useful feature of using Bloom filter for private set intersection is that the computation can be outsourced to an oblivious service provider, e.g. in cloud computing. The goal of an oblivious service provider – as opposed to the trusted third party in the ideal model – is that the provider does not learn anything about the inputs or the intersection (including its size). We modify our protocol as follows. The client still submits its encrypted Bloom filter b for its set $\{c_1, \ldots, c_v\}$, but the server now submits a Bloom filter b'_i for each s_i. Note that the service provider does not have access to the hash function results $h_j(s_i)$. The service provider

C	:	$b = Create(m)$
		$i = 1, \ldots v : b.Add(c_i)$
C $\longrightarrow$ P	:	$n, u, E_1(b_1), \ldots, E_1(b_m)$
S	:	$i = 1, \ldots, w : b'_i = Create(m)$
		$b'_i.Add(s_i)$
S $\longrightarrow$ P	:	$E_1(b'_{1,1}), \ldots, E_1(b'_{w,m}), E_2^l(s_1), \ldots, E_2^l(s_w)$
P	:	$i = 1, \ldots, w :$
		$j = 1, \ldots, m : E_2(\neg b_j \wedge b'_{i,j}) = \hat{e}(E_1(1) \cdot E_1(b_j), E_1(b'_{i,j})$
		$E_2^l(s'_i) = E_2^l(s_i) \times \prod_{j=1}^{m} Expand(E_2(\neg b_j \wedge b'_{i,j}) \cdot E_2(1))$
P $\longrightarrow$ C	:	$E_2^l(s'_1), \ldots, E_2^l(s'_w)$

Figure 4: Outsourced Private Set Intersection Protocol $OPSI$

obliviously to the server's set evaluates the *Test* function (on the ciphertexts) as

$$\bigwedge_{j=1}^{m} \neg(\neg b_j \wedge b'_{i,j})$$

The problem is that this formula has multiplicative depth 2 which can no longer be evaluated using the SYY technique. We therefore construct a new combination of homomorphic encryption system that can evaluate this formula and may be of independent interest. We use Boneh, Goh, Nissim (BGN) encryption instead of GM encryption which enables evaluating one multiplication of fan-in 2 (complementing unbounded addition). Then after evaluating the first logical-and we use again the SYY technique for the second logical-and.

5.1 Boneh, Goh, Nissim Encryption

Instead of describing the BGN encryption system as in [4], we will describe its simplification based on the techniques of GM encryption [19]. We emphasize that while our simplification reduces the plaintext domain from $\mathbb{Z}_q$ to $\mathbb{Z}_2$, it also has a few advantages. First, it is no longer necessary to solve the hard problem of discrete logarithm for decryption, but instead simple computations suffice. Second, the domain of the plaintext does not need be kept secret by the private key holder. Instead any party can perform operations, such as negation, in the group of the plaintext.

5.1.1 Cryptographic Pairings

Given a security parameter κ, let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two groups of order $n = pq$ for two large primes p and q, where the bit-size of n is determined by the security parameter κ. Our scheme uses a computable, non-degenerate bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. Modified Weil or Tate pairings on supersingular elliptic curves are examples of such maps. Recall that a bilinear pairing satisfies the following three properties:

- Bilinear: for $g, h \in \mathbb{G}_1$ and for $a, b \in \mathbb{Z}_n^*$, $\hat{e}(g^a, h^b) = \hat{e}(g, h)^{ab}$
- Non-degenerate: $\hat{e}(g, g) \neq 1$ is a generator of $\mathbb{G}_2$
- Computable: there exists an efficient algorithm to compute $\hat{e}(g, h)$ for all $g, h \in \mathbb{G}_1$

5.1.2 Algorithm

Recall that GM encryption is based on quadratic residues. In the first stage we encrypt using elements in group $\mathbb{G}_1$. We encode a 0 as a quadratic residue in $\mathbb{G}_1$, and a 1 as a pseudo quadratic residue in $\mathbb{G}_1$. Let $E_1(x)$ denote the encryption of plaintext x in this first stage and let $D_1(c)$ denote the corresponding decryption. Clearly, $E_1(x)$ is homomorphic in the exclusive-or operation.

$$D_1(E_1(x) \cdot E_1(y)) = x \oplus y$$

Using the properties of the bilinear map we can then also perform one logical-and operation on two ciphertexts. Since the result of the bilinear map inherits the exponents, if one operand is a quadratic residue, then the result will be a quadratic residue. The result will be either a quadratic residue or quadratic non-residue in $\mathbb{G}_2$. We then have a second-stage encryption scheme following the GM construction. Let $E_2(x)$ denote the encryption of plaintext x in this second stage and let $D_2(c)$ denote the corresponding decryption.

$$D_2(\hat{e}(E_1(x), E_1(y))) = x \wedge y$$

And again, the second-stage encryption system is homomorphic in the exclusive-or operation.

$$D_2(E_2(x) \cdot E_2(y)) = x \oplus y$$

We can now use the SYY technique on the second-stage encryption for unbounded fan-in logical-and. Let $E_2^l(x)$ denote the expanded ciphertext (using the same *Expand* algorithm as in Section 3.2).

$$D_2^l(E_2^l(x) \times E_2^l(y)) = x \wedge y$$

Using this combination of homomorphic encryption systems we can evaluate the formula above on ciphertexts.

5.1.3 Security

We adopt the IND-CPA security model for our construction. Our encryption algorithm does not involve any new primitives, but is a simplification of BGN encryption to GM encryption. We therefore make the following proposition.

PROPOSITION 1. *If the quadratic residuosity assumption holds, then our encryption scheme is IND-CPA secure.*

5.2 Protocol

In the outsourced protocol we introduce a third party, the service provider P. Both, client and server, send their encrypted inputs to the service provider who does not learn anything about either inputs or output. Figure 4 shows the protocol.

The protocol is only secure in the semi-honest model. Security against a malicious service provider – while not technically difficult – requires verifying the integrity of the computation which has the same complexity as performing the computation. Verifiable computation [7, 16] may help by shifting this complexity to a pre-computation phase, but is prohibitively expensive in practice due to its use of fully homomorphic encryption. We therefore argue that security in the malicious model annihilates any advantage of outsourcing.

Outsourced computation can be performed independently from either the client's or server's on-line availability. The client can store an encrypted Bloom filter at the service provider and the server may query new sets or elements as they arrive. Vice-versa, the server may store encrypted element-wise Bloom filter and bit-wise encrypted elements and the client may query using new Bloom filter.

5.2.1 Security Proof

The security of this protocol (in the semi-honest model) also trivially follows from the security of the encryption scheme.

In case of outsourcing we have to also simulate the view of the service provider – an additional party in the ideal model. We consider the service provider as just another party in the protocol with no input or output, i.e. its view is empty in the ideal model. It does not interact with the trusted server.

THEOREM 3. *If the quadratic residuosity assumption holds, then protocol OPSI implements private set intersection in the semi-honest model.*

PROOF. The server receives no messages and the client only the intersection, such that simulators are trivial.

In the protocol execution the service provider only receives encrypted messages, such that a simulator only needs to simulate ciphertexts. These are all independent due to IND-CPA security of our encryption scheme. □

6. CONCLUSIONS

We have investigated private set intersection protocols. We have presented a novel construction based on Bloom filter and homomorphic encryption. Our construction is secure against malicious adversaries in the standard model. Furthermore, we have the client certify its set with a trusted third party and hide the size of the client set from the server. Our construction is efficient. It has not only optimal complexity, but also constants similar to protocols only secure in the random oracle model. We extend our construction to outsourcing the computation of the intersection to an oblivious service provider. This makes private set intersection amenable to secure cloud computing.

7. REFERENCES

[1] G. Ateniese, E. De Cristofaro, and G. Tsudik. (If) Size Matters: Size-Hiding Private Set Intersection. *Proceedings of the 14th International Conference on Practice and Theory in Public Key Cryptography*, 2011.

[2] S. Bellovin, and W. Cheswick. Privacy-Enhanced Searches Using Encrypted Bloom Filters. *Cryptology ePrint Archive Report 2004/022*, 2004.

[3] B. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communication of the ACM 13(7)*, 1970.

[4] D. Boneh, E. Goh, and K. Nissim. Evaluating 2-DNF Formulas on Ciphertexts. *Proceedings of the 2nd Theory of Cryptography Conference*, 2005.

[5] J. Camenisch, and G. Zaverucha. Private Intersection of Certified Sets. *Proceedings of the 13th International Conference on Financial Cryptography and Data Security*, 2009.

[6] D. Chaum. Blind Signatures for Untraceable Payments. *Proceedings of CRYPTO*, 1983.

[7] K. Chung, Y. Kalai, and S. Vadhan. Improved Delegation of Computation Using Fully Homomorphic Encryption. *Proceedings of CRYPTO*, 2010.

[8] J. Coron. On the Exact Security of Full Domain Hash. *Proceedings of CRYPTO*, 2000.

[9] D. Dachman-Soled, T. Malkin, M. Raykova, and Moti Yung. Efficient Robust Private Set Intersection. *Proceedings of the 7th International Conference on Applied Cryptography and Network Security*, 2009.

[10] E. De Cristofaro, J. Kim, and G. Tsudik. Linear-Complexity Private Set Intersection Protocols Secure in Malicious Model. *Proceedings of ASIACRYPT*, 2010.

[11] E. De Cristofaro, Y. Lu, and G. Tsudik. Efficient Techniques for Privacy-Preserving Sharing of Sensitive Information. *Proceedings of the 2nd International Conference on Trust and Trustworthy Computing*, 2011.

[12] E. De Cristofaro, and G. Tsudik. Practical Private Set Intersection Protocols with Linear Complexity. *Proceedings of the 14th International Conference on Financial Cryptography and Data Security*, 2010.

[13] U. Feige, A. Fiat, and A. Shamir. Zero-Knowledge Proofs of Identity. *Journal of Cryptology 1(2)*, 1988.

[14] M. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword Search and Oblivious Pseudorandom Functions. *Proceedings of the 2nd Theory of Cryptography Conference*, 2005.

[15] M. Freedman, K. Nissim, and B. Pinkas. Efficient Private Matching and Set Intersection. *Proceedings of EUROCRYPT*, 2004.

[16] R. Gennaro, C. Gentry, and B. Parno. Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. *Proceedings of CRYPTO*, 2010.

[17] E. Goh. Secure Indexes. *Cryptology ePrint Archive Report 2003/216*, 2003.

[18] O. Goldreich. Foundations of Cryptography: Basic Applications. *Cambridge University Press*, 2004.

[19] S. Goldwasser, and S. Micali. Probabilistic Encryption. *Journal of Computer and Systems Science 28(2)*, 1984.

[20] C. Hazay, and Y. Lindell. Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries. *Proceedings of the 5th Theory of Cryptography Conference*, 2008.

[21] C. Hazay, and K. Nissim. Efficient Set Operations in the Presence of Malicious Adversaries. *Proceedings of the 13th International Conference on Practice and Theory in Public Key Cryptography*, 2010.

[22] S. Jarecki, and X. Liu. Efficient Oblivious

Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. *Proceedings of the 6th Theory of Cryptography Conference*, 2009.

[23] S. Jarecki, and X. Liu. Fast Secure Computation of Set Intersection. *Proceedings of the 7th International Conference on Security and Cryptography for Networks*, 2010.

[24] F. Kerschbaum. Public-Key Encrypted Bloom Filters with Applications to Supply Chain Integrity. *Proceedings of the 25th IFIP WG 11.3 Conference on Data and Applications Security*, 2011.

[25] L. Kissner, and D. Song. Privacy-Preserving Set Operations. *Proceedings of CRYPTO*, 2005.

[26] L. Michael, W. Nejdl, O. Papapetrou, and W. Siberski. Improving Distributed Join Efficiency with Extended Bloom Filter Operations. *Proceedings of the 21st International Conference on Advanced Networking and Applications*, 2007.

[27] M. Naor and B. Pinkas. Oblivious Transfer and Polynomial Evaluation. *Proceedings of the 31st Symposium on Theory of Computer Science*, 1999.

[28] R. Nojima, and Y. Kadobayashi. Cryptographically Secure Bloom-Filters. *Transactions on Data Privacy 2*, 2009.

[29] L. Qiu, Y. Li, and X. Wu. Preserving Privacy in Association Rule Mining with Bloom Filters. *Journal of Intelligent Information Systems 29(3)*, 2009.

[30] S. Ramesh, O. Papapetrou, and W. Siberski. Optimizing Distributed Joins with Bloom Filters. *Proceedings of the 5th International Conference on Distributed Computing and Internet Technology*, 2009.

[31] M. Raykova, B. Vo, S. Bellovin, and T. Malkin. Secure Anonymous Database Search. *Proceedings of the ACM Cloud Computing Security Workshop*, 2009.

[32] T. Sander, A. Young, and M. Yung. Non-Interactive CryptoComputing For NC1. *Proceedings of the 40th Symposium on Foundations of Computer Science*, 1999.

[33] A. Yao. Protocols for Secure Computations. *Proceedings of the IEEE Symposium on Foundations of Computer Science*, 1982.