Private Membership Test for Bloom Filters

Tommi Meskanen
University of Turku
Turku, Finland
tommi.meskanen@utu.fi

Jian Liu
Aalto University
Espoo, Finland
jian.liu@aalto.fi

Sara Ramezanian
University of Turku
Turku, Finland
sara.ramezanian@utu.fi

Valtteri Niemi
University of Helsinki
Helsinki, Finland
valtteri.niemi@helsinki.fi
and
University of Turku
Turku, Finland

***Abstract*—We study the problem of running a set membership test in private manner. We require that a client wants to have the option of not revealing the item for which the membership test is done. Respectively, the server does not wants to reveal the contents of the whole set. A Bloom filter is applied in the membership test. We present two protocols based on prior work as well as a new protocol. Each of these is having a slightly different privacy and complexity properties. We motivate the problem in the context of an anti-malware client checking application fingerprints against a cloud-based malware signature database.**

I. Introduction

With the widespread deployment of the Internet, increasingly more services are able to be provided to clients. This enables service providers to collect and exploit clients' personal information, including their interests, preferences, behaviors, and lifestyles. Service providers can collect such information easily because of the clear text nature of queries. Such collecting activities help providers increase the effectiveness and accurateness of their online advertising.

On the other hand, the collected information is open to further exploitation, which may leak clients' privacy. Seneviratne et al. find that a single snapshot of installed applications can reveal a lot information about user traits, e.g. religion, relationship status, spoken languages, countries of interest, and whether or not the user is a parent of small children [24] .

Membership test (or *set inclusion*) is a basic operation to access the remote data sources. It involves two main parties: a server who holds a database containing a set of records, a client who sends queries to test if a record is in the database. For example, the server can be hosted by a security company who holds a malware database. The client is an end-user who wants to install an application in her device and wants to know whether it is a malware. The client can simply send the name of the application to the server to test if it is in the database. But this leads to the above problem: the server can get extra information about the client by inspecting her installed applications. The problem can be solved by having the server send the whole database to the client. But this leads to high bandwidth usage, and is infeasible especially when the server doesn't want to reveal its database.

Another application of membership test is credit check. A seller can check with the financial services corporation whether a client has had any problems in his payment history in a such way that the name of the client is not revealed.

Private membership test (PMT) protocols enable clients to do membership test without revealing their queries and also prevent them from learning anything else about the server's database. Designing an efficient PMT protocol is a natural question in the field of secure computation.

In this paper we present three solutions for this problem. All of these protocols transfer minimal amount of data between parties. All three are also reasonably fast. Each of them have slightly different privacy properties. This makes it impossible to say that one of them is better than others in every respect. They all fit to slightly different situations with different requirements. One of the protocols is novel and one is only slightly modified from a protocol published earlier. The third one has been modified in order to make it more secure. The three protocols and their comparison are the main contributions of the paper.

We first introduce the preliminaries needed to understand this paper in Section II. Then, we formalize the problem in Section III. Next, we discuss the related work in Section IV. Then, we introduce three solutions for the PMT problem in Section V and give a comparison for them in Section VI. In the comparison we pay attention to the efficiency and study also the privacy properties in the light of our motivating scenario. Finally, we conclude the paper.

II. Preliminaries

A. Bloom Filter

Bloom filters are probabilistic data structures that can be used to test efficiently whether an element is a member of a set [4]. Specifically, a Bloom filter is an m-bit array B initialized with 0s, together with l independent hash functions $H_i(\cdot)$ whose output is uniformly distributed over $[0, m-1]$. To add an element x to the filter, one needs to feed it to all l hash functions to get l array positions ($h_i = H_i(x)$ for all $1 \leq i \leq l$), and then set all these positions of B to 1 ($B[h_i] = 1$). To check if an element is in the set, l positions are calculated in the same way. If any of these positions in B is 0, the element is not in the set. Otherwise, the element is *probably* in the set. The false positive rate depends on the value of m and l:

978-1-4673-7952-6/15 $31.00 © 2015 IEEE
DOI 10.1109/Trustcom.2015.414

$$FP_{bf} = (1 - (1 - \frac{1}{m})^{nl})^l \approx (1 - e^{-\frac{nl}{m}})^l, \quad (1)$$

where n is the size of the set i.e. the number of records stored in the Bloom filter. Based on this formula optimal values for m and l can be found when the size of the set and the wanted false positive rate are fixed.

B. Goldwasser-Micali Homomorphic Encryption

Goldwasser and Micali [13] proposed the first probabilistic encryption scheme in 1982, which is as follows:

- Gen is the key generation algorithm which generates two distinct large prime numbers p and q. Let $N = pq$ and y be a quadratic non-residue modulo N such that Jacobi$(y, N) = 1$. The tuple (N, y) is the public key pk and the tuple (p, q) is the private key sk.
- Enc is the encryption algorithm. Given a bit x, it outputs $c = y^x r^2 \mod N$, where r is a random element of $\mathbb{Z}_N^*$.
- Dec is the decryption algorithm. For each c, it outputs

$$x = \begin{cases} 0 & \text{if c is a quadratic residue modulo } N \\ 1 & \text{otherwise} \end{cases}.$$

The Goldwasser-Micali (GM) encryption scheme has the homomorprhic property. Given two ciphertexts $Enc(x_1)$ and $Enc(x_2)$ that are encrypted under the same key, one can calculate $Enc(x_1) \cdot Enc(x_2) = (y^{x_1} r_1^2) \cdot (y^{x_2} r_2^2) = y^{(x_1+x_2)} \cdot (r_1 r_2)^2 = Enc(x_1 + x_2) \mod N$.

C. Blind Signatures

Digital signature is a cryptographic scheme that enables a sender to demonstrate the authenticity of a message, and gives a recipient reason to believe that the message was created by the right party and was not altered during transmission .

Chaum introduced the notion of *blind signature* (BS) in 1983, which is the same as a normal digital signature scheme but the content of a message is blinded before being signed [6]. It involves two parties: a user Alice who holds a message x, and a signer Bob who holds a secret signing key sk. At the end of the protocol, Alice gets the signature without revealing anything about x and without learning anything about sk. Bellare et al. proposed a BS scheme based on the RSA assumption [2]. The scheme is shown in Figure 1. We denote the ideal functionality of BS as $(Sig(sk, x), \perp) \leftarrow \mathcal{F}_{BS}$ (x, sk).

Blind Signature

Alice		**Bob**
Input: N, e, x		Input: N, d
$r \xleftarrow{R} \mathbb{Z}_N^*$		
$x' \leftarrow r^e \cdot x \mod N$		
	$\bar{m}$ $\longrightarrow$	
		$x'' \leftarrow (x')^d \mod N$
	$\bar{x}$ $\longleftarrow$	
$Sig(sk, x)$ $\leftarrow r^{-1} \cdot x'' \mod N$		

Figure 1. A blind signature scheme based on RSA assumption [2].

D. Oblivious Pseudorandom Function

Pseudorandom functions (PRF) are efficiently-computable functions that are computationally indistinguishable from a *random oracle* (whose outputs are completely at random) [12]. Unlike *pseudorandom generators* which require their input seeds to be chosen at random, PRFs always output random values no matter how the inputs were chosen. Naor et al. point out that a PRF can be constructed based on the Decisional Diffie-Hellman assumption. Let p be the order of group G and let $\bar{r}$ be a vector of n values $(r_1, ..., r_n)$ sampled uniformly at random in $\mathbb{Z}_p^*$. For any n-bit $x = x_1 x_2 ... x_n$, the pseudorandom function $f_{\bar{r}}(x)$ can be constructed as $g^{\prod_{i=1}^n x_i r_i}$, where g is a generator of G [19].

Oblivious PRF (OPRF) are secure two party protocols that realize the ideal functionality $(f_k(x), \perp) \leftarrow \mathcal{F}_{OPRF}$ (x, k). Specifically, Alice holds a value x and Bob holds a key k. They jointly compute a pseudorandom function $f_k(x)$ without revealing their inputs. Freedman et al. construct an OPRF protocol which realizes Naor's PRF [9]. The protocol is based on *1-2 oblivious transfer*, which enables a sender to transfer one of two messages to a receiver without knowing which message has been transferred, and also without revealing the other message $((v_b, \perp) \leftarrow \mathcal{F}_{OT}$ $(b, \{v_0, v_1\}))$. Freedman's OPRF protocol is shown in Figure 2.

III. Problem Statement

We define the private membership test (PMT) protocol as the following functionality: The server $\mathcal{S}$ holds a set of n records $X = \{x_1, ..., x_n\}$. A query from a client $\mathcal{C}$ is a searchword x. The protocol outputs 1 to $\mathcal{C}$ if $x_i \in X$; otherwise, it outputs 0. The PMT protocol should guarantee the privacy for both parties: $\mathcal{S}$ should not learn anything about x and $\mathcal{C}$ should not learn anything else about X except the result of the protocol.

In our motivating scenario, $\mathcal{S}$ holds a database of malware or fingerprints of malware and $\mathcal{C}$ wants to check whether an application x she is going to install is in the database. Both of them want to protect their inputs, but $\mathcal{C}$ may reveal x to $\mathcal{S}$ to get more information if she finds x is a malware. It may also be in the Server's interest to be able to show that he cannot learn anything about x unless $\mathcal{C}$ explicitly reveals x to him.

For efficiency reasons we want to apply Bloom filters in the membership test.

The following numbers could be used as a typical baseline: There are 2^{21} records in $\mathcal{S}$'s database. Using formula (1) we find out that the false positive rate is acceptable (1:1000) when we use a Bloom filter of length 2^{25} and 10 hash functions are used to store the records.

Note that in this scenario, it is better to make error on the safe side, i.e. the problem caused by false positives is less

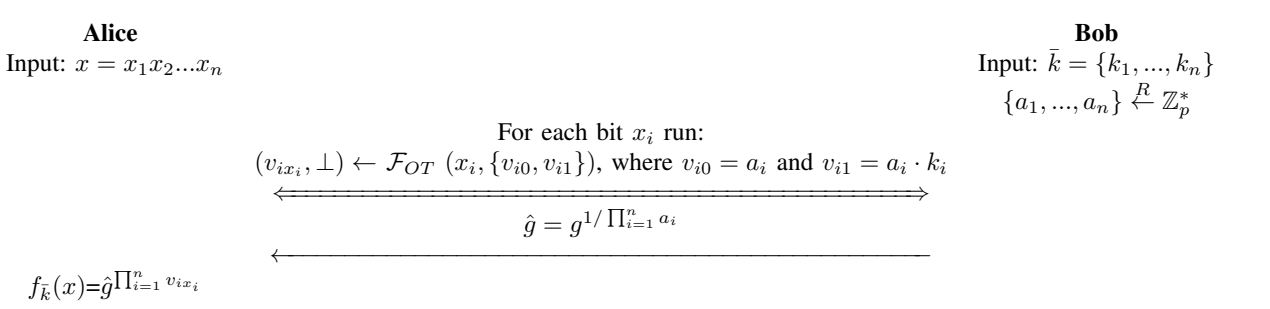

Figure 2. An oblivious pseudorandom function protocol [9].

serious than the problem of false negatives. Therefore using Bloom filters suits this scenario well.

To mitigate the problem of false positive results the following modification could be applied: In the case that the client gets a positive answer to his query, the client has the option of checking with the server that the result is correct by revealing his application to the server. This could also be done because the client wants to get more information about the malware.

IV. Related Work

A similar topic is called *private information retrieval* (PIR) which is well-known in cryptography. It assumes that $\mathcal{C}$ knows the index of her desired item in $\mathcal{S}$'s database and it enables $\mathcal{C}$ to retrieve the item without $\mathcal{S}$ being able to learn any information about which particular item was retrieved. The first PIR scheme was proposed by Chor et al. in 1995 [8]. Their scheme works in the scenario where there are replicated databases held by independent $\mathcal{S}$s. Kushilevitz and Ostrovsky introduced the first singer-server PIR scheme in 1997 [17]. After that, plenty of schemes are proposed in academia [5], [18], [11], [1].

Assuming that $\mathcal{C}$ knows the index of the item she wants to retrieve is not always realistic. So another topic called *private keyword search* (PKS) has been widely studied as well. In PKS, $\mathcal{S}$ holds a database of n pairs $\{(x_1, p_1), ..., (x_n, p_n)\}$, where x_i is a keyword and p_i is a payload. A query from $\mathcal{C}$ is a searchword x instead of an index. After the protocol, $\mathcal{C}$ gets the result p_i if there is a value i for which $x_i = x$ and obtains a special symbol $\perp$ otherwise. PKS can be constructed based on PIR [7], oblivious polynomial evaluation [9], re-routable encryption [23] or multiparty computation [21].

Any PKS algorithm can be turned into a PMT algorithm by setting all payloads to be 1 and returning 0 instead of $\perp$. Conversely, a PMT algorithm could be turned into a PKS algorithm by running it $\max\{|p_i|\}$ rounds and revealing one bit of the payload each time. In general fast PKS algorithms are slower than fast PMT algorithms and the amount of data transferred is larger. See for example [21].

Another linked topic is private set intersection and one of the recent papers on this topic is [25].

Unlike PIR and PKS, people pay less attention to the problem of PMT. Pinkas et al. propose an OT extension based private set inclusion protocol, which is used as a building block for their private set intersection protocol [22]. However, the communication complexity in their protocol is high ($\mathcal{O}(n)$ for each query). Nojima and Kadobayahi propose a privacy-preserving variant of Bloom filters that also meets the requirements of PMT. They propose two constructions based on BS and OPRF respectively [20]. Figure 3 shows the BS based construction and Figure 4 shows the OPRF based construction.

V. The protocols

In this section we provide three different solutions for the problem of PMT.

Protocol 1

Our first solution is novel and it uses Goldwasser-Micali homomorphic encryption as a building block. It is pictured in Figure 5.

Encrypting the Bloom filter: $\mathcal{S}$ first generates an *encrypted Bloom filter* EB as follows:

1) $\mathcal{S}$ stores X in B using l hash functions $H_1, \ldots, H_l$.
2) $\mathcal{S}$ chooses the parameter $N = pq$ for Goldwasser-Micali and another hash function H: $\{0,1\}^* \to \mathbb{Z}_N$.
3) For every index i in the Bloom filter, $\mathcal{S}$ finds by trial-and-error method the smallest j such that Jacobi$(H(j||i), N) = 1$.
   - If $H(j||i) \in \mathrm{QR}_N$ then $EB(i) = B(i)$.
   - If $H(j||i) \in \mathrm{QNR}_N$ then $EB(i) = 1 - B(i)$.
4) $\mathcal{S}$ sends to $\mathcal{C}$ hash functions H_i, $i = 1, \ldots, l$, the encrypted database EB, hash function H, modulus N and some $y \in \mathrm{QNR}_N$, such that Jacobi$(y, N) = 1$.
5) $\mathcal{S}$ is prepared to prove, if needed, to a trusted third party $\mathcal{TTP}$ (or to $\mathcal{C}$) that N is chosen as in Goldwasser-Micali and that $y \in \mathrm{QNR}_N$.

Querying the Bloom filter: For every hash function H_i, $i = 1, \ldots, l$.

1) $\mathcal{C}$ computes $H_i(x)$ and finds by trial-and-error method the smallest j such that Jacobi$(H(j||H_i(x)), N) = 1$.

Private Membership Test based on Blind Signature

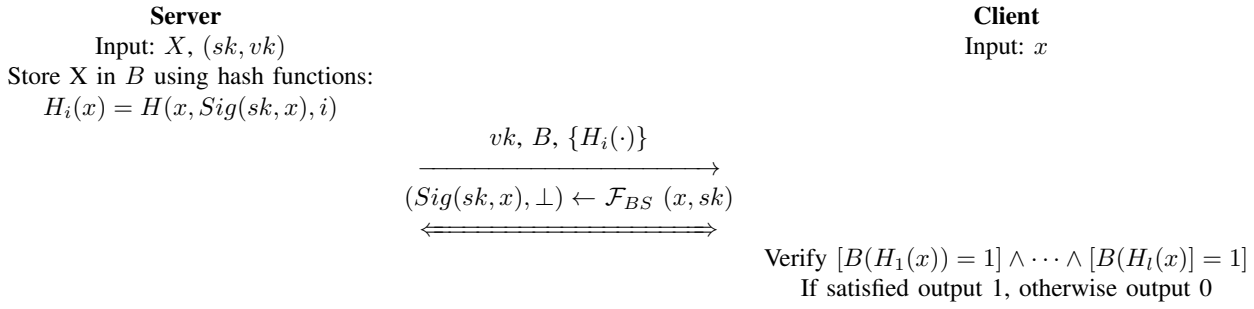

Figure 3. A private membership protocol based on blind signature [20].

Private Membership Test based on Oblivious Pseudorandom Function

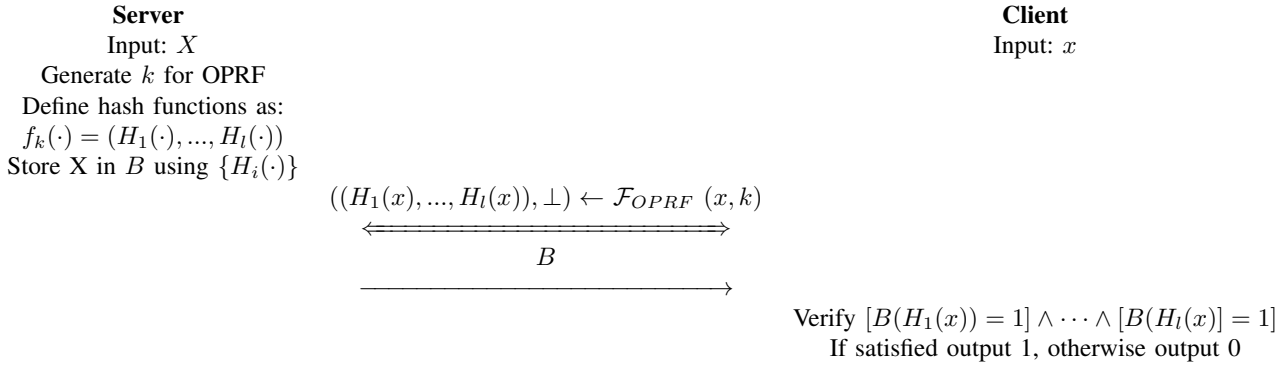

Figure 4. A private membership protocol based on oblivious pseudorandom function [20].

2) $\mathcal{C}$ multiplies $H(j||H_i(x))$ by a random square (mod N). He also multiplies it by y with probability $\frac{1}{2}$.
3) $\mathcal{C}$ sends the result z to $\mathcal{S}$.
4) $\mathcal{S}$ tells if this number is in QR_N or in QNR_N. The number z is a quadratic residue iff Jacobi(z, p) = Jacobi$(z, q) = 1$.
5) Now $\mathcal{C}$ is able to do the following reasoning:
   - If z is in QR_N and $\mathcal{C}$ did not multiply by y in step 2, he knows that $B(H_i(x)) = EB(H_i(x))$.
   - If z is in QNR_N and $\mathcal{C}$ did not multiply by y he knows that $B(H_i(x)) = 1 - EB(H_i(x))$.
   - If z is in QR_N and $\mathcal{C}$ did multiply by y he knows that the $B(H_i(x)) = 1 - EB(H_i(x))$.
   - If z is in QNR_N and $\mathcal{C}$ did multiply by y he knows that the $B(H_i(x)) = EB(H_i(x))$.

If for every hash function H_i, $\mathcal{C}$ got $B(H_i(x)) = 1$ then x is in the Bloom filter.

The size of the encrypted Bloom filter is equal to the size of the Bloom filter. This amount of data is transferred to $\mathcal{C}$. If the size of the Bloom filter is 2^{25} the amount of data needed to transfer is 4 MB. The content of EB is encrypted so there is no need to further encrypt it. It can be stored in a public database as long as the integrity of data is taken care of.

Note that because the probability that Jacobi$(r, N) = 1$ is $\frac{1}{2}$ for random value r, encrypting the Bloom filter requires in average two evaluations of Jacobi symbols for each index. Evaluating one Jacobi symbol has the time complexity $\mathcal{O}(\log(N)^2)$.

$\mathcal{S}$ can prove to a $\mathcal{TTP}$ that N is a product of exactly two primes using the algorithm of Gennaro et al. [10]. $\mathcal{S}$ can prove that y is a QNR_N using a zero-knowledge proof [14]. This way $\mathcal{S}$ can prove that he did not cheat choosing the parameters in the algorithm. The same database is typically used by several clients so it may be enough that $\mathcal{S}$ proves that parameters are correct only to one $\mathcal{TTP}$.

For every x, $\mathcal{C}$ has to evaluate in average $2l$ Jacobi symbols and multiply modulo N $2.5l$ times.

$\mathcal{C}$ sends to $\mathcal{S}$ one element of $\mathbb{Z}_N$ for each hash function H_i, l in total. $\mathcal{S}$ responds with l bits.

$\mathcal{S}$ needs to evaluate $2l$ Jacobi symbols for each query of x.

$\mathcal{C}$ learns the value of l positions in the filter, not only whether they are all ones. This is not what we originally wanted, but there are also other ways to find out individual bits in the Bloom filter: Lets assume that $\mathcal{C}$ knows $l-1$ positions in the Bloom filter with value 1 and he can choose any l positions in the filter such that he can find out using some protocol if

Private Membership Test based on Goldwasser-Micali Encryption

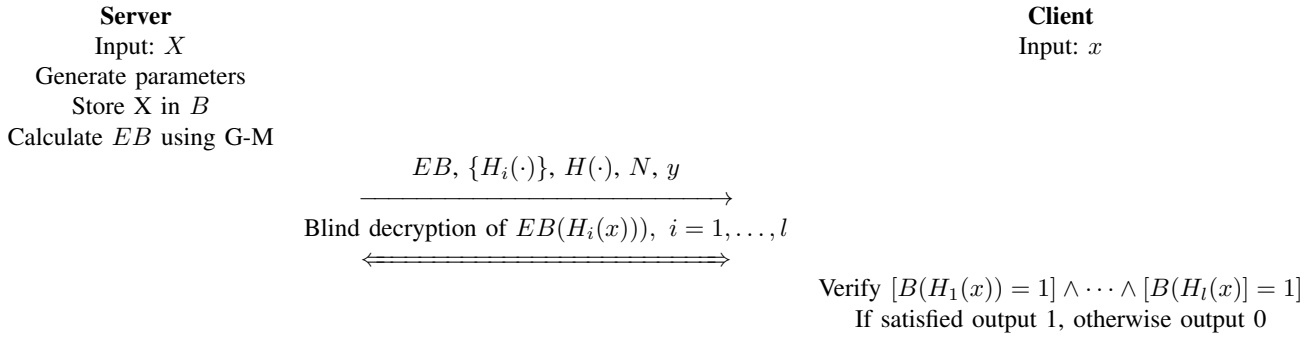

Figure 5. Protocol 1.

the value in all these positions is 1. In this situation he can always learn one more bit in the Bloom filter with each query.

The protocol has the following properties:

Theorem 1: The Client learns at most l bits of information about the bits in the Bloom filter.

Proof: If we assume that using the Goldwasser-Micali encryption and the hash function H is secure in this setting then $\mathcal{C}$ only gets to know the answers of $\mathcal{S}$ on step 4. Each of the l answers gives $\mathcal{C}$ at most one bit of information about the Bloom filter. ■

Theorem 2: The Server does not learn what positions the Client queries or what is the result.

Proof: Because $\mathcal{C}$ multiplies $H(j||H_i(x))$ by a random square on step 2, $\mathcal{S}$ does not recognize these values any more. Any quadratic residue can be turned into any quadratic residue by multiplying it by an appropriate square. Similarly, any quadratic non-residue with Jacobi value 1 can be turned into any quadratic non-residue with Jacobi value 1 by multiplying it by an appropriate square. If all squares are equally probable the result gives no information about what was the original quadratic residue (or quadratic non-residue with Jacobi value 1).

Because $\mathcal{C}$ multiplies $H(j||H_i(x))$ by a non-residue y with probability $\frac{1}{2}$ on step 2, $\mathcal{S}$ does not know if $H(j||H_i(x))$ was a quadratic residue or not. ■

$\mathcal{S}$ can change the encryption of the database at any time by choosing a different H or N. This way nobody has time to learn all the values of the Bloom filter before all values are changed to new ones.

Protocol 2

This protocol is adapted from the paper by Nojima et al. [20] and is pictured in Figure 3. We use blind RSA signature as the blind signature scheme. We have separated the two parts, generating the Bloom filter and querying it.

Generating the Bloom filter for signed records:

1) The Server $\mathcal{S}$ chooses keys e, d, N for RSA signature scheme. He also chooses l hash functions H_i for the Bloom filter and one hash function H to use with RSA signatures.
2) Let $Sig(x)$ be the signature for x, $Sig(x) = H(x)^d \pmod N$. $\mathcal{S}$ builds a Bloom filter B such that $B(h) = 1$ iff $h = H_i(x||Sig(x))$ for some hash function H_i and for some record x.
3) $\mathcal{S}$ delivers to $\mathcal{C}$ the Bloom filter B, hash functions $H_i, i = 1, \ldots, l$ and H, and his public RSA key (e, N).

Querying the Bloom filter: The Client $\mathcal{C}$ wants to know if x is stored in the Bloom filter of $\mathcal{S}$.

1) $\mathcal{C}$ picks a random $r \in \mathbb{Z}_N$ and calculates $y = r^e H(x) \pmod N$. He sends y to $\mathcal{S}$.
2) $\mathcal{S}$ signs y and returns the signature $z = y^d \pmod N$ to $\mathcal{C}$.
3) $\mathcal{C}$ calculates the signature of x, $Sig(x) = z/r \pmod N$.
4) If for every hash function H_i: $B(H_i(x||Sig(x))) = 1$ then $\mathcal{C}$ knows that x can probably be found in the database of records.

In order to generate the Bloom filter $\mathcal{S}$ needs to produce one RSA signature for every record in the database.

For every query both $\mathcal{S}$ and $\mathcal{C}$ need to do l exponentiations modulo N.

Because of the usage of blind signatures, $\mathcal{S}$ does not learn even the hash value of the Client's x.

$\mathcal{C}$ learns which l bits in the Bloom filter were relevant to x and their values. However, if he does not know $Sig(x')$ he does not know which entries to check if he wants to know if some x' is in the database or not.

$\mathcal{C}$ learns how many bits in the Bloom filter are ones because the filter is not encrypted. This number could be turned into a good estimate of the size of the database.

Protocol 3

The last protocol is more general and is based on OPRFs. This is also partly based on another protocol by Nojima et al. [20]. We have added the feature that the Bloom filter is encrypted using function K to hide the total number of

Our version of Private Membership Test based on Oblivious Pseudorandom Functions

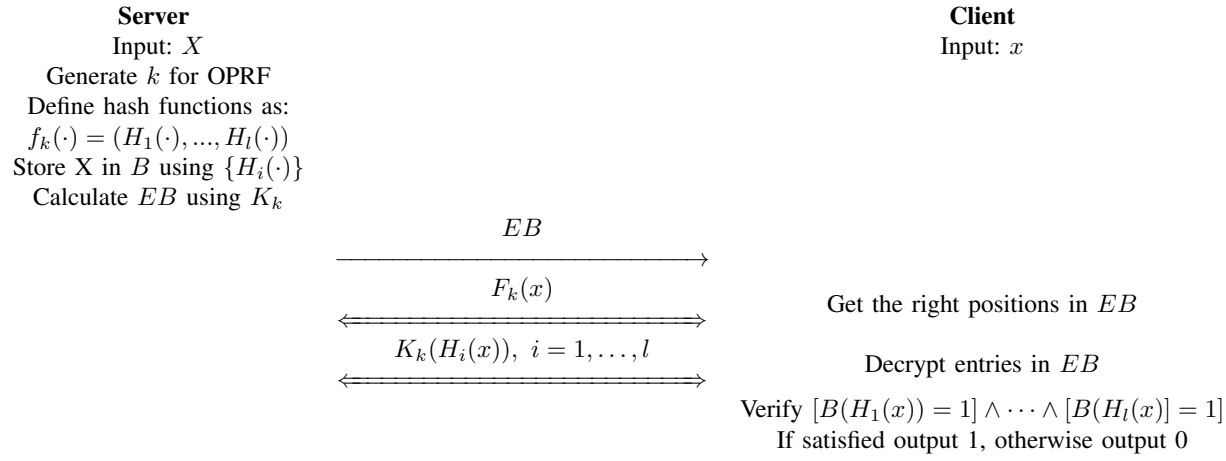

Figure 6. Protocol 3.

ones in the Bloom filter. This results in additional steps for generating the Bloom filter and decrypting the Bloom filter entries. Nojima et al. did not try to hide the number of ones in the Bloom filter. The protocol is pictured in Figure 6.

Generating the Bloom filter:

1) $\mathcal{S}$ picks a hash function H, his secret key k and hash functions $H_i,\ i = 1, \ldots, l$ for the Bloom filter.
2) $\mathcal{S}$ picks an OPRF K that can be evaluated using multiparty computation to generate one bit of output to be used as one-time pad key.
3) $\mathcal{S}$ picks another OPRF F that can be evaluated using multiparty computation such that
$$F_k(H(x)) = (H_1(x), H_2(x), \ldots, H_l(x)).$$
4) $\mathcal{S}$ builds a Bloom filter B such that $B(h) = 1$ iff $h = H_i(x)$ for some hash function H_i and record x.
5) $\mathcal{S}$ builds an encrypted Bloom filter EB such that $EB(i) = B(i) \oplus K_k(i)$ for every i.
6) $\mathcal{S}$ delivers the encrypted Bloom filter EB and the hash function H to $\mathcal{C}$.

Finding the right Bloom filter entries:

1) $\mathcal{S}$ and $\mathcal{C}$ together evaluate $F_k(H(x))$ such that only $\mathcal{C}$ learns the result $(h_1, h_2, \ldots, h_l)$, $\mathcal{C}$ does not learn k and $\mathcal{S}$ does not learn $H(x)$.

Decrypting the Bloom filter entries:

1) For every $h_i,\ i = 1, \ldots, l$, $\mathcal{S}$ and $\mathcal{C}$ together calculate $K_k(h_i)$ such that only $\mathcal{C}$ learns the result $b_i \in \mathbb{Z}_2$, $\mathcal{C}$ does not learn k and $\mathcal{S}$ does not learn h_i.
2) If $EB(h_i) \oplus b_i = 1$ for every i then $\mathcal{C}$ learns that x can probably be found in the database of records.

The size of the encrypted Bloom filter is again the size of the original Bloom filter.

For one query to the Bloom filter $\mathcal{C}$ and $\mathcal{S}$ must together evaluate F once and K l times.

VI. Comparison

Protocol 3 is the most general of these protocols. If $\mathcal{S}$ chooses the function F such that $\mathcal{C}$ can calculate it without his help, the resulting protocol resembles a lot of Protocol 1. Now the l hash functions H_i are public so $\mathcal{S}$ is only needed to decrypt the bits in the encrypted Bloom filter, just like in the Protocol 1. However, Protocol 1 is faster and may have fewer interactions than modified Protocol 3, depending on how the multiparty encryption function K is chosen.

On the other hand, if $\mathcal{S}$ chooses the value of function K to be 0 for all inputs (instead of using an OPRF), $\mathcal{S}$ is only needed for deciding which position of the Bloom filter to check. Now this resulting protocol resembles Protocol 2. Evaluating an OPRF typically requires several modular multiplications and this will make modified Protocol 3 slower than Protocol 2.

Protocol 1 reveals the hash functions of the Bloom filter and could eventually reveal the whole Bloom filter in bit-by-bit fashion. In Protocol 2 the Bloom filter is not encrypted and anybody can count the ones in the filter and convert the result into an estimation on how many different records it recognizes.

In Protocol 3 the hash functions and the number of ones in the Bloom filter remain secret. Even after repeated use nobody can decide if some x can be found in the database without the help of $\mathcal{S}$.

In Protocol 1, if $\mathcal{C}$ knows beforehand that $B(h) = 0$ and for some x and some i the value of $H_i(x) = h$, then he knows without contacting $\mathcal{S}$ that x is not in the database. The probability of this happening in practice is very small. If $\mathcal{C}$ knows 5 000 entries in the Bloom filter that are zeros, probability that at least one of $H_i(x)$ is among them is

$$1 - \left(1 - \frac{5000}{2^{25}}\right)^{10} \approx \frac{5000 \cdot 10}{2^{25}} \approx 0.15\%$$

when $l = 10$ and x is not in the database.

Instead of using an OPRF, it is possible to use garbled circuits. Using JustGarble [15] algorithm the running time of evaluation of one block of AES128 is 264 μs on an x86-64 Intel Core i7-970 processor clocked at 3.201 GHz with a 12MB L3 cache [3]. Garbling, or encrypting the function (in this case the AES128 function) and the argument (in this case the plaintext), takes 637 μs using the same machine. We could use this AES128 with a fixed key as a replacement of OPRF. Now Server's input to the computation is the fixed key which is held secret. The Client's input to the computation is the plaintext that will be encrypted. Comparing to using OPRF we have now the disadvantage that it may be required that $\mathcal{S}$ proves to a $\mathcal{TTP}$ that he has garbled a correct function. Furthermore, the Client and the Server need to use an OT protocol to transfer the garbling key for x. This will take much more time than the actual garbling and evaluation.

We assume that, as before, there are 2^{21} records in $\mathcal{S}$'s database and a Bloom filter of length 2^{25} and 10 hash functions are used to store the records.

Depending on l, we would need to evaluate AES128 different number of times. It is possible to interpret the output of 128 bits as five hash functions, each having an output of 25 bits. If $l \leq 5$ then one evaluation is enough and if $l \leq 10$ then two evaluations are enough to calculate the value of F on step 3. One AES128 is always enough to calculate the one bit of K on step 2.

Preprocessing the Bloom filter by $\mathcal{S}$ requires $2 \cdot 2^{25}$ Jacobi symbol evaluations in Protocol 1, one signature for each of the 2^{21} records in Protocol 2, and only evaluating the hash functions in Protocol 3.

The complexity of each query can be calculated as follows:

- In Protocol 1, each query requires $2l$ (in our scenario 20) Jacobi symbol evaluations by $\mathcal{C}$ and $2l$ Jacobi symbol evaluations for arguments of half of that size by $\mathcal{S}$.
- In Protocol 2, each query requires one modular exponentiation from both $\mathcal{C}$ and $\mathcal{S}$.
- In Protocol 3, each query requires one evaluation of F and l (in our scenario 10) evaluations of K from both $\mathcal{C}$ and $\mathcal{S}$ in cooperation.

These numbers are collected to Table I.

We have implemented protocols 1 and 2 using an x86-64 Intel Core i5-2450 processor clocked at 2.50 GHz with a 3MB L3 cache. The modulus in RSA and in Goldwasser-Micali was 2048 bits long.

Preprocessing i.e. generating the encrypted Bloom filter took about 6.9 hours in Protocol 1 and 27 hours in Protocol 2.

One query took time 0.05 seconds in Protocol 1 and 0.12 seconds in Protocol 2.

If we have $K = 0$ for all inputs we can use the results by Kreuter et al. [16] to approximate the speed of a query in Protocol 3. It took them 5.4 seconds to generate and evaluate an AES128 circuit twice using a 2.53 GHz Intel Core i5 processor and 4GB 1067 MHz DDR3 memory.

VII. Conclusion

We have presented three different solutions to the private membership test when using Bloom filters. The choice of a protocol depends on the privacy and efficiency requirements in the particular setting.

If it is not acceptable that somebody could save some queries by collecting entries in the Bloom filter in the case that the hash functions are known, then Protocol 1 should not be used. If anybody can evaluate the hash functions and lots of bits in the Bloom filter are known, it may happen that the Server is not needed to check what the relevant entries are in the Bloom filter.

If it is not acceptable that the number of ones in the Bloom filter is revealed then Protocol 2 should not be used. This may be the case because the number of ones in the Bloom filter can be turned into a good estimate of the number of records in the database.

The speed of JustGarble is based on particular hardware crypto accelerator features on certain processors and using it here also requires a fast oblivious transfer algorithm. If we do not have such hardware or any fast implementation of an OPRF then Protocol 3 should not be used.

Also the time complexities of the protocols vary. Depending on how fast evaluating a Jacobi symbol is compared to modular exponentiation and evaluating an OPRF, one of the protocol can be much faster than the others.

Our implementation shows that Protocol 1 is the fastest one and Protocol 3 is the slowest. However, Protocol 3 requires no preprocessing in addition to building the Bloom filter.

Acknowledgments

The authors would like to thank Professor N. Asokan for his valuable comments.

This work was supported in part by the Academy of Finland project "Cloud Security Services" (283135).

References

[1] C. Aguilar-Melchor and P. Gaborit. A lattice-based computationally-efficient private information retrieval protocol. *Crypto. ePrint Arch., Report*, 446, 2007.

[2] Bellare, Namprempre, Pointcheval, and Semanko. The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. *Journal of Cryptology*, 16(3):185–215, 2003.

[3] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 478–492, Washington, DC, USA, 2013. IEEE Computer Society.

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

[5] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In J. Stern, editor, *Advances in Cryptology EUROCRYPT 99*, volume 1592 of *Lecture Notes in Computer Science*, pages 402–414. Springer Berlin Heidelberg, 1999.

[6] D. Chaum. Blind signatures for untraceable payments. In D. Chaum, R. Rivest, and A. Sherman, editors, *Advances in Cryptology*, pages 199–203. Springer US, 1983.

[7] B. Chor. *Private information retrieval by keywords*. Citeseer, 1997.

[8] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, Nov. 1998.

[9] M. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In J. Kilian, editor, *Theory of Cryptography*, volume 3378 of *Lecture Notes in Computer Science*, pages 303–324. Springer Berlin Heidelberg, 2005.

Protocol	Preprocessing by $\mathcal{S}$	Query for $\mathcal{S}$	Query for $\mathcal{C}$	Privacy issues
Protocol 1	2^{26} Jacobi symbols	20 Jacobi symbols	20 Jacobi symbols	Some queries possible without $\mathcal{S}$
Protocol 2	2^{21} signatures	1 signature	1 mod. exp.	Reveals the size of database
Protocol 3		1 F (and 10 K's)	1 F (and 10 K's)	Slow in malicious model

Table I
SUMMARY OF COMPLEXITIES USING DIFFERENT PROTOCOLS

[10] R. Gennaro, D. Micciancio, and T. Rabin. An efficient non-interactive statistical zero-knowledge proof system for quasi-safe prime products. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, CCS '98, pages 67–72, New York, NY, USA, 1998. ACM.

[11] C. Gentry and Z. Ramzan. Single-database private information retrieval with constant communication rate. In L. Caires, G. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *Automata, Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pages 803–815. Springer Berlin Heidelberg, 2005.

[12] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, Aug. 1986.

[13] S. Goldwasser and S. Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 365–377, New York, NY, USA, 1982. ACM.

[14] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, Feb. 1989.

[15] JustGarble. http://cseweb.ucsd.edu/groups/justgarble/. Accessed: 19. March 2015.

[16] B. Kreuter, A. Shelat, and C.-H. Shen. Billion-gate secure computation with malicious adversaries. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 14–14, Berkeley, CA, USA, 2012. USENIX Association.

[17] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, 0:364, 1997.

[18] H. Lipmaa. An oblivious transfer protocol with log-squared communication. In J. Zhou, J. Lopez, R. Deng, and F. Bao, editors, *Information Security*, volume 3650 of *Lecture Notes in Computer Science*, pages 314–328. Springer Berlin Heidelberg, 2005.

[19] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2):231–262, Mar. 2004.

[20] R. Nojima and Y. Kadobayashi. Cryptographically secure bloom-filters. *Trans. Data Privacy*, 2(2):131–139, Aug. 2009.

[21] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. Keromytis, and S. Bellovin. Blind seer: A scalable private dbms. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 359–374, May 2014.

[22] B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on ot extension. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, pages 797–812, Berkeley, CA, USA, 2014. USENIX Association.

[23] M. Raykova, B. Vo, S. M. Bellovin, and T. Malkin. Secure anonymous database search. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, pages 115–126, New York, NY, USA, 2009. ACM.

[24] S. Seneviratne, A. Seneviratne, P. Mohapatra, and A. Mahanti. Predicting user traits from a snapshot of apps installed on a smartphone. *SIGMOBILE Mob. Comput. Commun. Rev.*, 18(2):1–8, June 2014.

[25] M.-E. Wu, S.-Y. Chang, C.-J. Lu, and H.-M. Sun. A communication-efficient private matching scheme in client-server model. *Inf. Sci.*, pages 348–359, 2014.

APPENDIX A: NOTATION TABLE

A table of notations is shown in Table II.

Notation	Description
Entities	
$\mathcal{C}$	Client
$\mathcal{S}$	Server
$\mathcal{TTP}$	Trusted Third Party
X	Server's database
x	Client's searchword
Cryptographic Notations	
PMT	Private membership test
H	Hash function
h	Hash value
B	Bloom filter
EB	Encrypted Bloom filter
QR_N	The set of quadratic residues modulo N
QNR_N	The set of quadratic non-residues modulo N
Sig	Signature
BS	Blind signature
$\mathcal{F}_{BS}$	Ideal functionality of blind signature
$OPRF$	Oblivious Pseudorandom Function
F	An OPRF to replace hash functions
K	An OPRF to calculate keys for EB
k	Key
Parameters	
l	Number of hash functions of the Bloom filter
m	Length of the Bloom filter
n	Number of records stored in the Bloom filter Number of records in the database

Table II
SUMMARY OF NOTATIONS