Controlling Data Disclosure in Computational PIR Protocols

Ning Shang
Purdue University
West Lafayette, IN, USA
nshang@cs.purdue.edu

Gabriel Ghinita
Purdue University
West Lafayette, IN, USA
gghinita@cs.purdue.edu

Yongbin Zhou
Purdue University
West Lafayette, IN, USA
yongbin@purdue.edu

Elisa Bertino
Purdue University
West Lafayette, IN, USA
bertino@cs.purdue.edu

ABSTRACT

Private Information Retrieval *(PIR)* protocols allow users to learn data items stored at an untrusted server, without disclosing to the server the particular data elements retrieved. Several PIR protocols have been proposed, which provide strong guarantees on the privacy of users. Nevertheless, in many application scenarios it is important to protect the database as well. For instance, the data may represent an important asset for the server, and billing is often performed with respect to the number of items retrieved. When applying PIR protocols in such scenarios, both computation/communication efficiency and database/user privacy must be considered. However, most prominent efficient computational PIR (cPIR) solutions have been designed to consider user privacy only. Furthermore, we show that the few symmetric PIR solutions that address server privacy are vulnerable to subliminal channel attacks. Therefore, devising PIR protocols that preserve user privacy, and at the same time control tightly the number of data items disclosed by the server remains an open problem.

In this paper, we investigate the amount of data disclosed by the the most prominent PIR protocols during a single run. We show that a malicious user can stage attacks that allow an excessive amount of data to be retrieved from the server. Furthermore, this vulnerability can be exploited even if the client follows the legitimate steps of the PIR protocol, hence the malicious request can not be detected and rejected by the server. We devise mechanisms that limit the PIR disclosure to a single data item. We have implemented our proposed protocol, and we show experimentally that the extensions that we operate do not incur significant performance overhead.

1. INTRODUCTION

Consider the case of a database server that stores information about diseases and their associated symptoms and publishes the associated indices for search. Alice, who is experiencing health problems, wishes to query the server in order to determine a candidate diagnosis. However, Alice wants to keep her health status private, so she does not want the server, or some malicious eavesdropper, to learn the contents of her query. Therefore, Alice should be able to find a possible diagnosis, without the server learning any of Alice's symptoms.

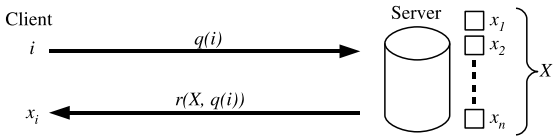


Figure 1: Overview of Computational PIR

Private Information Retrieval (PIR) addresses such application scenarios. The PIR problem was first formulated [CGKS95] in the context of binary data, represented as a bit string $X = [x_1, \ldots, x_n]$. The client[1] holds an index i and wishes to privately retrieve the value of bit x_i. In an information-theoretic setting (i.e., assuming that the adversary has unbounded computational power), it is shown in [CGKS95] that the required communication cost is $\Theta(n)$, if only a single database server exists. In other words, Alice needs to retrieve the entire database to protect her privacy. Obviously, such an approach is not practical. In order to reduce this bound, the authors of [CGKS95] propose a solution that replicates the database on a number of K non-colluding servers, and yields $O(K \log K \cdot n^{\frac{1}{\log K}})$ communication complexity. In a more recent work [BIKJF02], this bound was improved to $O(n^{\frac{\log \log K}{K \log K}})$. Nevertheless, the existence of several non-colluding servers may be a difficult assumption to meet in practice, from both security, as well as feasibility[2] considerations.

On the other hand, the assumption of an adversary with unbounded computational power may not be necessary in practice. In many situations, it is enough to guard against adversaries with polynomial-time computational capabilities. This version of the problem is

[1]We use the terms *user* and *client* interchangeably.

[2]Current business models are not compatible with an architecture where a service is jointly provided by several distinct (possibly competitor) parties.

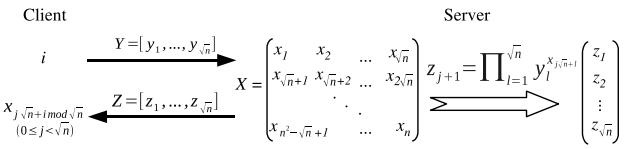


Figure 2: Overview of PIR protocol from [KO97]

	Client Privacy	Database Privacy	
		RA	SCA
[KO97]	Yes	No	No
[CMS99]	Yes	Yes	No
[NP99]	Yes	Yes	No
Our Approach	Yes	Yes	Yes

Table 1: Summary of PIR Approaches. RA: redundancy attack. SCA: subliminal-channel attack

called *Computational PIR (cPIR)*. cPIR is outlined in Figure 1: the client, who wants to find the value of the i^{th} bit of X (i.e., x_i), sends an encrypted request $q(i)$ to the server. The server responds with a value $r(X, q(i))$, which allows the client to determine the value x_i. It is computationally intractable for an adversary to find the value of i, given $q(i)$. Furthermore, the client can easily determine the value of x_i based on the server's response $r(X, q(i))$, using a trap-door function.

The work in [KO97] was the first to show that it is possible to achieve cPIR with communication sub-linear to the database size, even if the database is not replicated on multiple servers. The solution relies on the *Quadratic Residuosity Assumption (QRA)*. QRA states that it is hard to decide whether a random number is a quadratic residue or not [Fla88] with respect to a large composite modulus, and achieves $O(n^{\epsilon})$ communication complexity, where $\epsilon > 0$ is an arbitrarily-small positive constant. More recently [CMS99], this communication bound was improved to $O(\log^{\beta} n)$ (where $\beta > 1$ is a system parameter) by a solution that relies on the *ϕ-hiding assumption (ϕHA)*. ϕHA conjectures that it is computationally hard to decide whether a random prime p divides $\phi(m)$, where ϕ is the *Euler totient function* [Fla88] and m is a large composite number whose integer factorization is unknown. Note that, in order to prevent the server from learning which data item has been requested, the server-side component of *any* PIR protocol must process every element in the database. Hence, the computational complexity is at least linear to the data size. Nevertheless, it has been shown [GKK+08] that certain optimizations can reduce the cPIR overhead to a level that is reasonable for practical applications.

The primary goal of the PIR protocols discussed so far is to protect the privacy of the client. However, the database represents an asset for the server. In traditional database applications, clients are typically billed in proportion to the amount of transferred data. Designing an effective billing mechanism is an important concern with PIR, since the server does not learn which particular data item has been retrieved by the client. Nevertheless, the protocol design should control the amount of data (e.g., the number of elements) retrieved during a single PIR request. Otherwise, a malicious user could abuse the system.

Given practical concerns as above, in this paper, we identify two important classes of attacks that allow clients to retrieve excessive information with a single PIR request: *redundancy attacks (RA)* and *subliminal-channel attacks (SCA)*. The former class is relevant to protocols that disclose redundant data items due to their design. The latter class of attacks is more subtle, and applies to most cPIR protocols that we are aware of. We briefly illustrate these attack classes using the protocol from [KO97] (Figure 2). The server organizes the binary data array X as a $\sqrt{n} \times \sqrt{n}$ square matrix. The client request consists of an array Y of $\sqrt{n}$ large numbers, specifically chosen such that item i can be retrieved (the exact details of how the request is created are presented in Section 3.1). The server performs a masked product of numbers in Y based on the values of X. Specifically, a factor y_l is included in the product z_j (corresponding to matrix row j) only if the data item $x_{j\sqrt{n}+l}$ has value **1**. All z values are returned to the client, who checks each product z_{j+1} and learns the values of all $\{x_{j\sqrt{n}+i \bmod \sqrt{n}}\}$, $0 \le j < \sqrt{n}$ (i.e., all the values in the column $i \bmod \sqrt{n}$ of the data matrix). Note that, such disclosure is a result of redundant information being revealed to the client, which in turn is a consequence of the protocol design aiming to minimize the overall communication cost (i.e., the sum of sizes of arrays Y and Z). Therefore, this disclosure belongs to the RA class. In addition, a malicious client can stage a SCA attack by factorizing each product z_j, and learn additional information[3] about X values that do not belong to column $i \bmod \sqrt{n}$. In certain cases, a malicious client could learn the values of all items in the database. For instance, in our earlier example, Alice could retrieve the entire database of diagnoses with a single PIR request, and sell the data to a third party. Furthermore, the malicious requests abide the protocol specification, therefore it is virtually impossible for the database server to detect and filter such harmful queries.

The problem of private queries with database protection, or *Symmetric PIR (sPIR)*, has been studied previously in [GIKM98]. sPIR requires that the client only learns the value of the requested bit x_i, and no information about any other data item x_j ($j \neq i$). sPIR was introduced in an information-theoretic context [GIKM98], and a solution with multiple non-colluding servers was proposed. However, as mentioned earlier, the assumption of multiple non-colluding servers may be difficult to meet in practice. Database privacy was also considered in the cPIR setting [NP99, MS00, NR06]. The work in [NP99] is a meta-protocol that can be used in conjunction with any cPIR protocol to eliminate redundancy in the server response. In consequence, it is still vulnerable to subliminal channel attacks. On the other hand, [MS00, NR06] are designed as modifications of the protocol in [KO97] targeted to protect the database. However, the solution in [NR06] is flawed (as we show in Section 4), whereas the work in [MS00] incurs exorbitant cost that makes it unusable for any practical applications (to retrieve 1 bit of data in a database of size 1GB and with a reasonable security parameter of 1024, the communication cost is 4GB.[4]) Table 1 summarizes the characteristics of existing solutions with respect to client and server privacy. Note that, none of the existing techniques is able to solve both aspects of privacy (i.e., client and server), in the presence of both RA and SCA attacks.

The contribution of our work is two-fold:

[3]The details of the attack are presented in Section 4.
[4]Assume the data are organized in a square matrix.

1. We analyze the unintended data disclosures during the operation of the two most prominent cPIR protocols ([KO97, CMS99]), and we show that in the worst case the amount of disclosure is linear to the database size.

2. We propose modifications that protect server privacy against both RA and SCA attacks. Specifically, we focus on securing the protocol in [KO97], which has been shown previously [GKK+08] to obtain reasonable performance in practice. We implemented the revised protocol, and we present an experimental evaluation which shows that the proposed extension is scalable to large databases.

The rest of the paper is organized as follows: Section 3 reviews the details of the two cPIR protocols in [KO97, CMS99], whereas Section 4 illustrates the vulnerabilities of these protocols to RA and SCA attacks. In Section 5, we present the proposed solution that protects both client and server privacy, and we evaluate experimentally our technique in Section 6. Finally, Section 7 concludes the paper and identifies several interesting directions for future work.

2. RELATED WORK

Private Information Retrieval (PIR) allows a user to fetch the value of bit i from a server that owns a bit-string $x_1 \dots x_n$ of length n, without the server learning the value of i. A trivial solution is for the user to download the entire bit-string, but such an approach has communication cost $\Theta(n)$. A *non-trivial* PIR protocol has two requirements: (P1) protect user privacy, i.e., keep the value of i secret from the server, and (P2) incur sub-linear (i.e., $o(n)$) communication cost. Chor et al [CGKS95] were the first to introduce the PIR concept, and they showed that in an information-theoretic setting, there is no non-trivial solution with a single server. The same authors propose a scheme with $K \geq 4$ non-colluding servers with $O(n^{1/\log K} K \log K)$ communication cost. This bound was improved to $O(n^{\frac{\log\log K}{K \log K}})$ in [BIKJF02]. More recently, Yekhanin [Yek07] showed that if infinitely many Mersenne primes exist, then there is a three-server PIR protocol with $O(n^{1/\log\log n})$ communication complexity.

In practice, the assumption of multiple non-colluding servers may not often be met. Computational PIR (cPIR) is concerned with finding efficient single-server solutions that are robust against adversaries with polynomially-bounded computation capabilities. The seminal work of Kushilevitz and Ostrovski [KO97] proposed the first cPIR protocol that relies on the Quadratic Residuosity Assumption (QRA) and has $O(n^{\epsilon})$ communication cost for arbitrarily small $\epsilon > 0$. The bound was improved by Cachin et al [CMS99] who proposed a poly-logarithmic communication protocol that relies on the ϕ-hiding assumption (ϕHA). More efficient solutions have been proposed for a slight variation of PIR, namely Private Block Retrieval (PBR). In PBR, the user retrieves a block of bits, rather than a single bit. Lipmaa [Lip05] introduced a $O(\log^2 n)$ communication protocol, whereas Chang [Cha04] proposed a solution with $O(\log n)$ communication complexity based on the Paillier [Pai99] cryptosystem. Finally, Gentry et al [GR05] introduced a constant-rate, communication-efficient PBR protocol that can be used under several distinct intractability assumptions, including ϕHA.

A closely-related concept to PIR is *Oblivious Transfer (OT)*, originally introduced in [Rab81]. OT, also referred to as *1-out-of-n* transfer, has two requirements: (O1) the server should not learn the value of i, and (O2) the client should not learn the value of any bit other than the i^{th} bit. Note that, OT does not specify any condition on efficiency, as opposed to requirement (P2) of PIR. In fact, several OT protocols, e.g., [NP99], have $\Theta(n)$ communication complexity. *Symmetric PIR (sPIR)* was originally introduced in [KO00], and combines the benefits of OT and PIR: specifically, it protects the privacy of both the client and the server, and is communication efficient (i.e., incurs $o(n)$ communication cost). However, the solution in [KO00] assumes multiple non-colluding servers. The work in [CMO00] shows that there exists a communication-efficient reduction from any PIR protocol to a *1-out-of-n* OT protocol, which results in a sPIR protocol. Nevertheless, such an approach may still be vulnerable to subliminal-channel attacks, as we will show in Section 4. In contrast, our work eliminates subliminal channel inferences, and also proves experimentally the feasibility of our proposed sPIR approach for the single-server setting.

Finally, several works consider the relationship between PIR, OT and other cryptographic primitives. In [CMO00] it is shown that non-trivial PIR is complete, namely it can be used to construct any other protocol problem, whereas the work in [IL89] shows that OT implies the existence of one-way functions. On the other hand, the work in [CMO00] brings strong evidence that although the existence of one-way functions is necessary for non-trivial PIR, it is not a sufficient condition.

Notation	Description
n	Number of Data Objects
$X = [x_1, \dots, x_n]$	Server Database (array of bits)
QR/QNR	Set of quadratic residues/non-residues.
$N = q_1 \cdot q_2$	Composite modulus of bit length k
$t = \lceil \sqrt{n} \rceil$	Size of PIR Matrix in the KO protocol
$M[t \times t]$	PIR Matrix of bits in the KO protocol
$Y, Z[1, \dots, t]$	Query and Result arrays in the KO protocol

Table 2: Summary of Notations

3. REVIEW OF EXISTING CPIR PROTOCOLS

We review the two most prominent solutions for computational PIR. Section 3.1 presents the KO protocol [KO97], which is based on the *Quadratic Residuosity Assumption (QRA)*, whereas Section 3.2 reviews the CMS protocol [CMS99] which relies on the *ϕ-hiding assumption (ϕHA)*. To facilitate presentation, we summarize the notations used in the description of the protocols in Table 2.

3.1 QRA-based PIR

Consider large k-bit composite integer N (i.e., product of two $\frac{k}{2}$-bit primes q_1 and q_2), where k is a security parameter. We refer to N as *modulus*, and k as modulus bit-length. The Quadratic Residuosity Assumption (QRA) states that it is computationally hard to determine whether a given random number is a quadratic residue (QR) or a quadratic non-residue (QNR) modulo N, provided that the factorization of N is not known.

Let

$$\mathbb{Z}_N^* = \{x \in \mathbb{Z}_N | \gcd(N, x) = 1\},$$

be the set of positive numbers in $\mathbb{Z}_N$ which are relatively prime to N (gcd is the greatest common divisor). Then, the set of *quadratic residues (QR)* modulo N is defined as:

$$QR = \{y \in \mathbb{Z}_N^* | \exists x \in \mathbb{Z}_N^* : y = x^2 \pmod N\}. \quad (1)$$

The complement of QR in $\mathbb{Z}_N^*$ represents the set of *quadratic non-residues (QNR)*.

Let

$$\mathbb{Z}_N^{+1} = \{y \in \mathbb{Z}_N^* | \left(\frac{y}{N}\right) = 1\}, \quad (2)$$

where $\left(\frac{y}{N}\right)$ denotes the Jacobi symbol [Fla88]. Then, exactly half of the numbers in $\mathbb{Z}_N^{+1}$ are in QR, while the other half are in QNR. According to QRA, for a random $y \in \mathbb{Z}_N^{+1}$, it is computationally intractable to decide whether $y \in$ QR or $y \in$ QNR. Formally, define the quadratic residuosity predicate Q_N such that:

$$Q_N(y) = 0 \Leftrightarrow y \in \text{QR} \quad (3)$$

DEFINITION 1 (QUADRATIC RESIDUOSITY ASSUMPTION). *For every constant $\alpha > 0$ and any function $C(N, y)$ computable in polynomial time in k, there exists k_0 such that*

$$\forall k > k_0, \Pr_{N \in H_k, y \in \mathbb{Z}_N^{+1}}[C(y) = Q_N(y)] < \frac{1}{2} + \frac{1}{k^\alpha}, \quad (4)$$

where $N \in H_k, y \in \mathbb{Z}_N^{+1}$ denotes the experiment of first drawing two $k/2$-bit primes p_1, p_2 and compute $N = p_1 \cdot p_2$, and then drawing y uniformly randomly from $\mathbb{Z}_N^{+1}$.

Intuitively, the probability of distinguishing between a QR and a QNR is negligible for large-enough k.

The KO protocol organizes database X as a $t \times t$ square[5] matrix M, where $t = \lceil\sqrt{n}\rceil$, such that $M_{i,j} = x_{(i-1)\cdot t+j}$ (for ease of presentation, we assume that n is a perfect square; however, the protocol supports arbitrary values of n, in which case the M matrix can be padded with extra values).

When the client wishes to retrieve the data item at index i, it first generates two random $\frac{k}{2}$-bit primes q_1 and q_2 and computes the modulus $N = q_1 \cdot q_2$. Next, the client determines the row $a = \lceil i/t \rceil$ and the column $b = i \pmod t$ corresponding to x_i in matrix M, and assembles the query $q(i)$ which is a $1 \times t$ array

$$Y = [y_1, y_2, \ldots, y_t]. \quad (5)$$

Y consists of t random k-bit numbers such that $Q_N(y_b) = 1$, (i.e., y_b is in QNR), whereas all other elements $y_j, j \neq b$ are in QR. The query Y is sent to the server together with modulus N (note that, the factorization of N is kept secret).

When the server receives the query, it computes for each row r of M the product

$$z_r = \prod_{j=1}^{t} y_j^{M_{r,j}} \pmod N \quad (6)$$

and sends the resulting array

$$Z = [z_1\ z_2\ \ldots\ z_t] \quad (7)$$

back to the client. Note that, each z_r value is a k-bit number obtained by multiplying those y_j values for which the corresponding data bit $M_{a,b} = 1$ (i.e., a masked product). Therefore

$$M_{a,b} = \begin{cases} 1, & z_a \in \text{QNR} \\ 0, & z_a \in \text{QR} \end{cases}. \quad (8)$$

Knowing the factorization of N, the client can efficiently check whether or not $z_a \in$ QR, and learn the value of $M_{a,b}$. Note that, in addition to the requested item $M_{a,b}$, the client can use the remaining Z values in order to recover all elements in column b of M, i.e., a total of $\sqrt{n}$ data bits.

In [NR06], a modification to the KO protocol is proposed in order to reduce the amount of data disclosed to the client. Next, we show that this approach is flawed, and does not achieve the desired data protection outcome. The idea behind [NR06] is to apply (6) to matrix M in two separate stages, both row-wise and column-wise. However, the number of required communication rounds is not increased: the client creates an additional query array Y' such that y'_a is QNR, whereas all other elements of Y' are QR. The server computes two sets of replies, corresponding to Y and Y', and also includes a random component to the protocol. For each matrix element $M_{r,c}$, the server performs a coin-flip to obtain a random bit $\rho_{r,c}$ (either 0 or 1). Then, it computes

$$z_r = \prod_{c=1}^{t} (y_c \cdot y_c^{\rho_{r,c}})^{M_{r,c}} \pmod N \quad (9)$$

and

$$z'_c = \prod_{r=1}^{t} (y'_r \cdot y_r'^{\tilde{}\rho_{r,c}})^{M_{r,c}} \pmod N, \quad (10)$$

where $\tilde{}\rho_{r,c}$ is a bit flip of $\rho_{r,c}$. The server sends back to the client Z and Z' and the client multiplies the a^{th} element of Z with the b^{th} element of Z' to obtain the product $u = z_a \cdot z'_b$, from which it can be determined that

$$M_{a,b} = \begin{cases} 1, & u \in \text{QNR} \\ 0, & u \in \text{QR} \end{cases} \quad (11)$$

Eq. (11) is correct, because if $M_{a,b} = 1$, then exactly one of y_a and y'_b is included directly (i.e., not squared) as a factor in u, therefore $u \in$ QNR; and similarly, $M_{a,b} = 0$ if and only if $u \in$ QR. Furthermore, given $M_{a,b} = 0$, for any data bit $M_{r,c}$ such that either $r = a$ or $c = b$ but not both, no information can be derived by the client if $M_{r,c} = 0$ (the corresponding $z_r \cdot z'_c$ value will always be in QR). However, if $M_{r,c} = 1$, there is a non-negligible 50% chance that the resulting $z_r \cdot z'_c$ will be QNR (depending on the outcome of the coin flip for $\rho_{r,c}$ and $\rho'_{r,c}$), so the data disclosure is reduced only to half. A similar argument also holds when $M_{a,b} = 1$. Therefore the claim of the authors in [NR06] that redundancy attacks are completely prevented does not hold.

The computational complexity of KO is $O(n)$, since one multiplication of large integers is performed for all bits in X which have value 1. The communication complexity is $O(\sqrt{n})$, since each of the query Y and result Z have $t = \sqrt{n}$ elements. It is shown in [KO97] that the communication overhead can be reduced to $O(n^\epsilon)$, for arbitrary $\epsilon > 0$, by recursively applying the KO protocol on the result Z. However, for practical values of n, the additional computational overhead of the recursive version may not be justified.

3.2 ϕ-hiding-based PIR

The more recent work in [CMS99] proposes the CMS protocol which improves on the asymptotic communication cost of KO. Specifically, the $O(n^\epsilon)$ bound is reduced to $O(\log^\beta n)$ where $\beta > 1$ is a

[5]For brevity of presentation, we only consider square matrices. Nevertheless, the KO protocol, as well as the vulnerability we will present in Section 4.1, also apply to general rectangular (i.e., non-square) matrices.

system parameter.[6] CMS relies on the *ϕ-hiding assumption (ϕHA)*, which states that given large composite prime N, it is difficult to decide (without knowing the factorization of N) whether a small prime divides $\phi(N)$, where $\phi(N)$ is the *Euler totient function* at N (i.e., the number of integers smaller than N that are co-prime to N).

A prime number p is said to be *ϕ-hidden* by N if $p|\phi(N)$. Denote by $\mathcal{P}^b$ the set of b-bit primes: $H^b(N)$ is defined as the subset of $\mathcal{P}^b$ consisting of primes that are hidden by N, whereas $\bar{H}^b(N)$ is the complement of $H^b(N)$ with respect to $\mathcal{P}^b$. Let H_a^b denote the set of N, products of two a-bit primes, which ϕ-hide a b-bit prime.

Formally, the ϕ-hiding assumption states that:

DEFINITION 2 (ϕ-HIDING ASSUMPTION). *$\exists e$, f, g, $k_0 > 0$ such that $\forall k > k_0$ and $\forall 2^{ek}$-gate circuits $C(\cdot,\cdot)$,*

$$\Pr_{\substack{N\in H^k_{kf}\\ p_0\in H^k(N), p_1\in \bar{H}^k(N)\\ b\in\{0,1\}}}[C(N,p_b)=b] < \frac{1}{2}+2^{-gk}, \qquad (12)$$

where the probability is taken by first drawing N uniformly by random from H^k_{kf}, then drawing p_0 and p_1 uniformly by random from $H^k(N)$ and $\bar{H}^k(N)$, respectively, and flipping a coin for the value of bit b.

In other words, the probability of distinguishing in polynomial time whether N hides a given prime p or not is as good as guessing.

In the protocol, the client possesses a prime-sequence generator $\mathcal{PG}$ that computes a deterministic sequence of k-bit primes $p_1 \dots p_n$, one for each item in the database. The client also computes an integer N that hides the prime p_i, corresponding to the item x_i that the client wishes to retrieve. As shown in [CMS99], the probability of N hiding any other prime $p_j, j \neq i$ is negligible. The client also generates a random $g \in \mathbb{Z}_N^*$ and sends to the server g, the generator $\mathcal{PG}$ and N.

The server computes recursively the following:

$$\begin{aligned} v_0 &= g,\\ v_j &= v_{j-1}^{p_j^{x_j}} \pmod N, 1 \le j \le n \end{aligned} \qquad (13)$$

and returns to the client v_n. Note that each v_j is obtained by raising the previous v_{j-1} to a power equal to the prime $p_j^{x_j}$.

The client receives from the server the value v_n, and determines that

$$x_i = \begin{cases} 1, & \text{if } v_n \text{ has a } p_i^{\text{th}} \text{ root modulo } N\\ 0, & \text{otherwise} \end{cases}. \qquad (14)$$

This is correct because if bit i is 0, then the value of v_i is equal to v_{i-1}, and p_i has not contributed to the value of v_n; otherwise, an exponentiation with p_i was performed, and a p_i^{th} power was introduced. Since the client knows the factorization of N, determining whether a p_i^{th} root exists can be done by checking if $v_n^{\phi(N)/p_i} = 1 \pmod N$.

3.3 Comparison of PIR Protocols

The two PIR protocols discussed in this section rely on different intractability assumptions. Both protocols offer a similar level of protection for the client privacy. The KO protocol performs only modular multiplications of large integers, whereas CMS requires modular exponentiations with large primes. On the other hand, the order of magnitude of the p_j primes used by CMS is much lower than the modulus length of KO. Still, in practice, it is likely that KO would incur a significantly lower computational overhead than CMS. Furthermore, the KO protocol is deterministic, in the sense that it always returns the correct result to the client, as opposed to CMS which is probabilistic in nature (although the probability of returning the correct result is very high).

On the other hand, KO discloses to the client a number of $\sqrt{n}$ data items in each request (corresponding to a column of the PIR matrix). CMS is much better at protecting the database privacy, as it is designed to disclose a single data item per query. Nevertheless, as we show next in Section 4, both protocols are vulnerable to attacks that may lead to a larger fraction of the database being disclosed to a malicious client. In particular, with one single PIR request, the client could potentially retrieve the entire database in the case of KO, and considerably more than one item for CMS.

4. EXCESSIVE DISCLOSURE OF DATA IN PIR PROTOCOLS

In most applications, the database is a valuable asset to the server, and client billing is performed with respect to the number of data items retrieved. Therefore, the PIR protocol should enable the server to control the number of disclosed items, even if the server does not learn which exact items were retrieved. Ideally, only one single item (the object of the client request) should be revealed by the protocol. Next, we show how a malicious client can exploit the KO and CMS protocols to gain knowledge on a number of data items considerably larger than intended by the PIR protocols' design. In this section, we focus on SCA attacks only. The RA attack[7] for KO has been discussed in detail in Section 3.1.

4.1 Exploiting KO

As mentioned in Section 3.1, the KO protocol is designed to disclose no more than $\sqrt{n}$ data items in a single request. This is already a large fraction of the database. Still, a malicious client can stage an attack that can increase the data disclosure even further, by injecting a subliminal channel within the query message.

Consider that the dishonest client rewrites the query array y (Eq. (5)) such that y_j is the j^{th} prime in the sequence of prime numbers starting with $y_1 = 2$. The server will multiply these numbers according to Eq. (6), and obtain the array z of composite numbers, one for each row of matrix M. Since the y_j values are relatively small numbers, the client can easily recover from z_i the value of $M_{i,j}$, $1 \le j \le t$, by factorizing z_i. Specifically,

$$M_{i,j} = \begin{cases} 1, & \text{if } y_j | z_i\\ 0, & \text{otherwise} \end{cases}.$$

EXAMPLE 1. *Suppose $n = 16, t = 4$ and*

$$M = \begin{pmatrix} 1 & 1 & 0 & 1\\ 1 & 0 & 0 & 0\\ 0 & 1 & 1 & 0\\ 1 & 0 & 0 & 1 \end{pmatrix}$$

[6] According to the authors, a value of $\beta = 8$ is suitable in practice

[7] Recall that CMS is not vulnerable to RA attacks.

The client chooses $N = 1152921515344265237 = 1073741827 \times 1073741831$, *and*

$$Y = [2\ 3\ 5\ 7]$$

and sends them to the server. The server computes and returns to the client

$$Z = [42\ 2\ 15\ 14]$$

By factoring z_1, *the client obtains* $42 = 2 \times 3 \times 7$. *Hence, the client learns that the first row of* M *must be* $[1, 1, 0, 1]$. *The same process is repeated for all other values* z_i. □

Since the Y values considered so far are small and in increasing order, the server may use a filter that rejects queries consisting of such y_j value sequences. However, the client can further obfuscate the server by changing the order of y_j above according to a permutation δ. The client can also *cover* each y_j value by multiplying it with a random number $\rho \in Z_N^*$. After receiving z from the server, the client divides each z_i by $\rho^l, 1 \leq l \leq t$ (the client can perform division because it knows the factorization of N) until a relatively small number is obtained. By inspecting the factorization of this number, and knowing permutation δ, the client can reconstruct the values of all $M_{i,j}$. Example 2 below illustrates this case.

EXAMPLE 2. *The client can choose a cover*

$$\rho = 1059514023252037783,$$

and query the server with message

$$N = 1152921515344265237 = 1073741827 \times 1073741831$$

and

$$Y' = \rho \cdot Y = \begin{bmatrix} 966106531159810329 \\ 872699039067582875 \\ 685884054883127967 \\ 499069070698673059 \end{bmatrix}.$$

The server returns

$$Z = \begin{bmatrix} 1109565644447711234 \\ 966106531159810329 \\ 339122049744744175 \\ 700821084876516309 \end{bmatrix}$$

to the client. To obtain the 4^{th} *data row, the client divides* $z_4 = 700821084876516309$ *by* ρ, ρ^2, ρ^3, ρ^4 *modulo* N, *and obtains* $998138141397346118, 14, 231893657280329621, 911165485062388888$, *respectively. Notice that* $14 = 2 \times 7$ *is considerably smaller than the other values. The client can readily tell that the* 4^{th} *row of* M *is* $[1, 0, 0, 1]$. *By continuing the process, the client can recover the other values of data matrix* M. □

One key observation is that although it is believed to be hard in general to factor large integers, it is easy to do so when the integer to be factored is composed of factors from a chosen factor base. In the original KO protocol [KO97], if a client can carefully choose the query message in such a way that all y_j are distinct small prime numbers, and the product of all y_j is noticeably smaller than the modulus N, then the entire database from the server can be retrieved by a malicious client. Certainly, in practice, t is much larger than that in the above examples, and thus it is possible that the product of all the first t primes is larger than N. In this case, the client can choose the first $t' < t$ primes so that their product satisfies the desired property, assigns these t' primes to some of the y_j, and set the other y_j to be 1. In this way the client can retrieve t'/t of the database with a single query. For example, when $N \approx 2^{1024}$, t' can be chosen to be at least 100.

To sum up the attack, a malicious client can retrieve a considerably large portion of the entire database from the server with a single PIR request, by computing the factorization of each element in Z. Furthermore, it is hard for the server to detect whether a query message Y is malicious or not, because the Y values can be covered through multiplication with a random number. In fact, a malicious request can be formed such that one single element is QNR, because the client can always choose y_j as small numbers, exactly one of which is a QNR, then cover them by multiplying them with a QR modulo N.

4.2 Exploiting CMS

As shown in Section 3.2, the client can reconstruct the value x_i of data item i by checking if the server response v_n has a root of order p_i. In the following, we show how an attacker can retrieve more than one data item with a single request.

The CMS protocol considers that a client generates an integer N that hides only a single prime p_i out of the prime sequence $p_1 \ldots p_n$. However, the client could generate an N which hides a subset $p_{i_1} \cdots p_{i_\ell}$, and retrieve the values for all data items $i_1 \cdots i_\ell$. The challenging part of this attack consists of finding a value of N of suitable size which hides a larger fraction of primes p_i. Since revealing a large prime dividing $\phi(N)$ may compromise N's factorization, it is suggested in [CMS99] that only small p_i be chosen. However, it turns out that in this case it is easy to find such an N in practice. Indeed, we have the following Algorithm 1.

Algorithm 1 Generating N that ϕ-hides $p_1, \ldots, p_\ell$

Input: ℓ k-bit primes $p_1, \ldots, p_\ell$, and a constant $f > 1$
Output: an integer N that is a product of two k^f-bit primes and ϕ-hides p_i for $1 \leq i \leq \ell$.

1: Compute $q_1 \leftarrow \prod_{i=1}^{\ell_1} p_i, q_2 \leftarrow \prod_{i=\ell_1+1}^{\ell} p_i$, where $\ell_1 = \lceil \ell/2 \rceil$.
2: **repeat**
3: Choose a random integer h_1 of approximately $(k^f - \ell_1 \cdot k)$ bits long, and compute a k^f-bit integer $r = q_1 h_1 + 1$.
4: **until** r is prime.
5: **repeat**
6: Choose a random integer h_2 of approximately $(k^f - (\ell - \ell_1) \cdot k)$ bits long, and compute a k^f-bit integer $s = q_2 h_2 + 1$.
7: **until** s is prime.
8: Set $N \leftarrow r \cdot s$.

As with the analysis in [CMS99], under the Extended Riemann Hypothesis (cf., [BS96]), Algorithm 1 above generates a suitable N in expected polynomial time in $\ell \cdot k$. Examples 3 and 4 show in detail how the attack works.

EXAMPLE 3. *Consider* $k = 32$ *and* $f = 1.8$. *The following choice of primes allows the retrieval of* $\ell = 12$ *distinct data items*

in only one request:

$$\begin{aligned} p_1 &= 4294967311 \\ p_2 &= 4294967357 \\ p_3 &= 4294967371 \\ p_4 &= 4294967377 \\ p_5 &= 4294967387 \\ p_6 &= 4294967389 \\ p_7 &= 4294967459 \\ p_8 &= 4294967477 \\ p_9 &= 4294967497 \\ p_{10} &= 4294967513 \\ p_{11} &= 4294967539 \\ p_{12} &= 4294967543 \end{aligned} \tag{15}$$

The associated 1024*-bit* N *that hides all* $p_i, 1 \le i \le 12$ *is*

$$\begin{aligned} N = {} & 10112023883600526980977341698188264126601120506550474471630442065122463014059429176214851849387575842782354180452470707683073215429227255338286959652238538973035023914879926118517805961174161689837573468397741248527993013470924939335935085254676051428354916559582167693033488865257259098319471008049145614398 7 \\ = {} & r \times s \\ = {} & 10055855947456947824680518748654384595609524365444295033292671082791323022555160232601405723627212032346243055537474411534528645988754619344908986523292289 \times \\ & 10055855947456947824680518748654384595609524365444295033292671082791323022555160232601405723625283097674709787837582662070260937329355698931183594512356083. \quad \square \end{aligned}$$

Given that N ϕ-hides all $p_j, 1 \le j \le \ell$, a malicious client can query the database server with p_j as members of the prime sequence. Since the client knows the factorization of N, it can determine the value x_j (corresponding to the prime p_j) by checking whether the server's response has a p_j^{th} root, for all $1 \le j \le \ell$. In the following example of the CMS protocol, we show how a client may retrieve 12 bits within a single database query.

EXAMPLE 4. *Let* g*,* $p_1, \ldots, p_n$*, and* N *be part of the query message a client sends to the database server in a running CMS protocol. Suppose* $p_i, 1 \le i \le 12$*,* r*,* s *and* N *are as in Example 3 and* g *is randomly chosen from* $\mathbb{Z}_N^*$ *as*

$$\begin{aligned} g = {} & 39501796803163927427652757863283379545391135652903855067612584895278837244198860780741844283772487029416901132632325695760405098306291829875324952529081344432646000373420727766022512428696863516213083754989110454004326948615511861638050525137537505666866741289450363558792900931751595964707657620643894320792. \end{aligned}$$

Then N ϕ*-hides* p_i *for all* $1 \le i \le 12$*. Without loss of generality, we may assume* $n = 12$*. Suppose the database has data*

$$\begin{aligned} X = {} & (x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 0, x_5 = 0, x_6 = 1, \\ & x_7 = 1, x_8 = 0, x_9 = 0, x_{10} = 0, x_{11} = 0, x_{12} = 1). \end{aligned}$$

The server computes according to Eq. (13)

$$\begin{aligned} v_n = {} & 856755606989645043235435509738683970328496949225032228433878374987251374990809852633850757891814809896369938505971522900130776295917795925904823539158021519839109729280860833458334354749239566883559586445610393606983354760810079894514163061158650731262132480535226825084592335357977553980186425156221442437 4 \pmod N, \end{aligned}$$

and sends the client v_n*. Knowing the factorization of* N*, the client checks that*

$$\begin{aligned} x_i' = v_n^{\phi(N)/p_i} \pmod N &= v_n^{(r-1)\cdot(s-1)/p_i} \pmod N \\ &= 1 \pmod N, \end{aligned}$$

for $i = 2, 6, 7$ *and* 12*, and that* $x_i' \neq 1 \pmod N$ *for the other indices* i*. Hence the client retrieves all* 12 *bits in the database* X*.* $\square$

Without knowing N's factorization, the server is not able to tell whether N ϕ-hides any p_i, according to the ϕ-hiding assumption. Therefore it is hard for the server to distinguish a malicious query from an honest one.

5. A SECURE SYMMETRIC PIR PROTOCOL

In this section, we propose a single-database computational PIR protocol that ensures database protection by releasing one data item at one time, and is suitable for practical use. We chose to extend the QRA-based PIR technique from [KO97] since it is, to our knowledge, the only cPIR protocol that achieves good performance[8] in practice [GKK+08]. Our approach consists of two steps: first, in Section 5.1 we eliminate the disclosure due to returning redundant elements to the user. Next, in Section 5.2 we show how to defend against SCA attacks that use factorization to infer excessive information.

5.1 Protecting against RA Attacks

As discussed in Section 3.1, the KO protocol returns the user the entire array $Z = [z_1, \ldots, z_t]$, $t = \sqrt{n}$, where each element z_j corresponds to the product of the numbers in query array Y masked by data items in row j of matrix M. We aim to limit the disclosure to a single element z_j corresponding to the row of data item x_i requested by the client (i.e., $j = \lfloor i/t \rfloor$). For this purpose, we make use of a *1-out-of-*n oblivious transfer protocol proposed in [NP99].

Although both our proposed solution to single-database sPIR and the one suggested in [NP99], further referred to as NP, use a PIR and a *1-out-of-*n protocols in their constructions, the difference is that in our solution we use KO (PIR scheme) and *1-out-of-*n protocols in a serial way, whereas in NP a (generic) PIR scheme is used to replace one step of *1-out-of-*n. Both schemes can achieve a comparable communication complexity $O(\sqrt{n})$. However, whereas both schemes require $O(n \log n)$ evaluation of pseudo random functions and $O(n)$ modular multiplications, our scheme needs only

[8]The CMS protocol is considerably slower, since it performs n large integer exponentiations, versus n multiplications for KO.

$O(\sqrt{n})$ public-key operations, but NP will need $O(n)$ public-key operations, if a comparable implementation is adopted. Given that one public-key operation is a lot more expensive than a single modular multiplication in general, our proposed scheme is more suitable for practical applications. In Section 6 we implement our scheme, and provide experimental data to confirm this observation.

A simplistic solution for data protection is to use encryption together with a *1-out-of-n* oblivious transfer protocol. The server uses a family of pseudo random functions $\{F_K : \{0,1\}^m \rightarrow \{0,1\} | K \in \{0,1\}^s\}$, where m is the size of data in bits and s is a large enough key length (so that an exhaustive search is not possible) which we will see later. Each data element x_i has its own associated key K_i. In order to learn data item x_i, the client downloads the entire (encrypted) database, and then performs a *1-out-of-n* oblivious transfer (OT) protocol (which we will describe shortly) to learn the value of K_i. Since the client knows only one encryption key, she/he learns only one data item. Note that applying the *1-out-of-n* protocol directly on the entire database X is not a good solution, since the entire (encrypted) database is transferred to the client (i.e., communication cost is $\Theta(n)$). In our solution, we employ the *1-out-of-n* protocol as an extension of KO, and we encrypt the elements of the Z array. Since the original KO protocol sends the entire Z to the client in the first place, we do not incur any additional communication cost to that extent (as we will show shortly, the encrypted values have the same length as the plaintext ones). We do, however, incur more communication cost during the *1-out-of-t* protocol that retrieves the decryption key.

We provide the array $Z = [z_1, \ldots, z_t]$ as an input to the *1-out-of-t* protocol. The client needs to learn the value z_j, $j = \lfloor i/t \rfloor$, in order to reconstruct data item x_i. By only disclosing to the user the value of key K_j, we ensure that no additional data is learned by the user due to redundancy (i.e., redundancy attacks are thwarted). Denote by $\ell = \log t$ the number of bits necessary to encode any index value j, $1 \leq j \leq t$ and by $\overline{j_1 \ldots j_\ell}$ the binary representation of index value j. The server generates a set of ℓ pairs of s-bit keys (K_p^0, K_p^1), $1 \leq p \leq \ell$, and for each z_j it chooses the encryption key K_j as tuple $K_j = (K_1^{j_1}, \ldots, K_\ell^{j_\ell})$. Recall from Section 3.1 that each z element is a k-bit integer. The server uses the pseudo random function $F_K : \{0,1\}^\ell \rightarrow \{0,1\}^k$, and creates the ciphertexts $E_{K_j}(z_j)$, $1 \leq j \leq t$ as follows:

$$E_{K_j}(z_j) = z_j \oplus F_{K_1^{j_1}}(j) \oplus \cdots \oplus F_{K_\ell^{j_\ell}}(j) \qquad (16)$$

Note that, since encryption is performed solely through XOR operations, the ciphertexts and the plaintexts have equal length.

The client who wants to retrieve z_j, $j = \overline{j_1 \ldots j_\ell}$, downloads all encrypted Z values, and performs ℓ runs *1-out-of-2* OT protocols [NP99] to retrieve all the components $K_1^{i_1}, \ldots, K_\ell^{i_\ell}$, of key K_j. Note that distinct keys may share some components (if the binary representations of corresponding indices have the same value in a certain bit). However, all components are required for successful decryption, hence the scheme is secure. Furthermore, the sharing reduces the cost of the key retrieval, which only involves ℓ runs of the *1-out-of-2* protocol.

Figure 3 gives an overview of the proposed extension to KO. In Figure 3(a), the server encrypts each of the z_i values according to Eq. (16), whereas Figure 3(b) illustrates the key retrieval step. Note that, the two steps are shown separately for clarity of presentation. However, both steps can be performed in a single communication round, since the two phases are independent from each other. Furthermore, if the computation infrastructure allows it, the two steps can be executed in parallel.

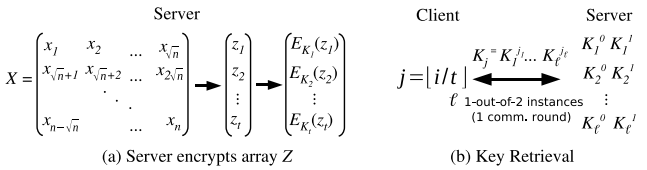


Figure 3: Eliminating Redundant Disclosure in the KO Protocol

To completely specify our extension, we describe how the *1-out-of-2* protocol works. The server has two messages m_0 and m_1, and the client has a bit $b \in \{0,1\}$. The client wishes to learn message m_b without letting the server find b, whereas the server wants to ensure that the client only learns one of the two messages. The following steps are performed:

1. The server chooses a public-key encryption algorithm (E, D), where the ciphertext space is the entire space of m-bit values, and two random m-bit values x_0 and x_1. Then it sends the client E, x_0 and x_1.
2. The client chooses a random value c and sends $q = E(c) \oplus x_b$ to the server.
3. The server computes $c_0 = D(q \oplus x_0)$, $c_1 = D(q \oplus x_1)$, $d_0 = m_0 \oplus c_0$, $d_1 = m_1 \oplus c_1$, and sends d_0, d_1 to the client.
4. The client computes $m_b = d_b \oplus c$.

The proposed extension of the KO protocol has two main cost components: encrypting each value in array Z, and performing ℓ runs of the *1-out-of-2* protocol. The *computational* complexity of encrypting Z is $O(t)$, whereas no additional communication overhead is incurred for transferring the encrypted Z to the client. The *1-out-of-2* protocol requires the server and the client each to perform two public-key operations for all bits of K_j (steps 2 and 3 above). The cost of these operations depends on the key length chosen by the server for the (E, D) pair. In the experiments, we will explore the effect of varying the server key size.

The required *1-out-of-2 communication* can be combined with the query-response process in the KO protocol. The only extra communication cost is incurred by sending random values x_0 and x_1 before the query-response process starts. In practice, these values can be piggybacked with packets from the server to the client when the network communication is initialized, before the actual PIR session executes. Therefore, the communication overhead introduced by applying *1-out-of-2* can be decreased in real-world applications.

5.2 Protecting against SCA Attacks

Note that the attack on the KO protocol, described in Section 4.1, relies on the fact that the adversary receives (or recovers, if a random cover is used) a z_i which is noticeably smaller than the modulus N. However, for a client to correctly retrieve the desired bit $M_{a,b}$ in the KO protocol, only the residuosity of the server response z_a is needed. Note that, 1) multiplying the number with a QR does

not change its residuosity, and 2) the server can easily generate a random QR (by squaring a random number) without needing to know the factorization of the modulus N. Therefore, to defend against the subliminal-channel attack, the server can *blind (multiply)* each response z_i with a random QR ρ_i modulo N, and return to the client

$$z'_i = \rho_i \cdot z_i \pmod{N}, \quad 1 \leq i \leq t.$$

All z'_i created in this way have the same residuosity as z_i, and with overwhelming probability are approximately the same size as the modulus N. This still allows an honest client to recover the desired bit by following the procedures instructed in the KO protocol, and at the same time prevents a malicious client from retrieving more bits via integer factorization. Since there are around $N/4$ QRs, each of which could be a blinding value, the malicious client cannot effectively unblind the retrieved value, hence is not able to reconstruct the subliminal channel as shown in Section 4.1 to learn additional data items.

EXAMPLE 5. *Let us continue the discussion in Example 1 with the above counter-measure patched on the server. Suppose the client queries the server with the same message as in Example 1. Instead of sending back the response $Z = [42\ \ 2\ \ 15\ \ 4]$ the server randomly chooses from $\mathbb{Z}_N$ ($N = 1152921515344265237$) blinding factors*

$$\rho_1 = 119951277712694732^2 \pmod{N} = 929831830491486306,$$
$$\rho_2 = 477296616356918650^2 \pmod{N} = 855518740578228759,$$
$$\rho_3 = 718908649135391281^2 \pmod{N} = 1005154117284639206,$$
$$\rho_4 = 899202885004181252^2 \pmod{N} = 259841459381492705,$$

computes

$$\begin{aligned} z'_1 &= \rho_1 \cdot z_1 \pmod{N} = 1006526874281672031 \\ &= 3 \cdot 29 \cdot 1087 \cdot 10643306731399, \\ z'_2 &= \rho_2 \cdot z_2 \pmod{N} = 558115965812192281 \\ &= 13 \cdot 37 \cdot 4099 \cdot 96587 \cdot 2930777, \\ z'_3 &= \rho_3 \cdot z_3 \pmod{N} = 89332059794140009 \\ &= 7 \cdot 11 \cdot 433 \cdot 2679345544349, \\ z'_4 &= \rho_4 \cdot z_4 \pmod{N} = 179015885308102159 \\ &= 23 \cdot 79 \cdot 199 \cdot 495089330273, \end{aligned}$$

and returns the client $Z' = [z'_1, z'_2, z'_3, z'_4]$. We show here the prime factorization of z'_i to indicate that it does not give the client any meaningful information to infer the bit values in the database. In practice, when the modulus N is big, the client may even not have the capability to factor large integers z'_i. Without knowing the blinding factors ρ_i, the injected subliminal channel in the query cannot be recovered by the client. □

The random blinding requires the server to generate a random square modulo N, and perform one more modular multiplication for each row of the data matrix M. The extra computational cost introduced by blinding is $O(n^{1/2})$, negligible compared to the original KO protocol which requires $O(n)$ modular multiplications. It is clear that blinding does not increase the communication cost.

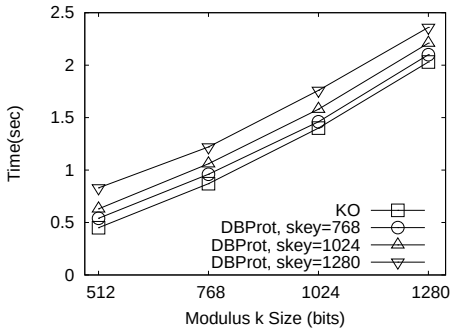
6. EXPERIMENTAL EVALUATION

We evaluate experimentally the proposed protocol for PIR with database protection (labeled *DBProt*). We developed a C++ prototype, using the GnuMP library[9] for manipulating large integers.

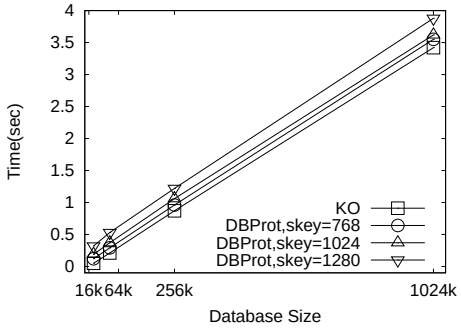
[9]http://www.gnump.org

We choose RSA as the public-key algorithm for the 1-out-of-2 OT protocol. The tests were run on a Pentium 4 3.0 GHz with 1GB of RAM running Linux 2.6 OS. We consider KO modulus bit lengths of $k = 512$, 768, 1024 and 1280 bits, and RSA server key sizes (for key pair (D, E)) of $skey = 768$, 1024 and 1280 bits. The database size is varied between 16K and 1024K elements (2^{14}, 2^{16}, 2^{18} and 2^{20} elements).

Fig. 4 shows the server execution time for DBProt in comparison with the original KO protocol (we omit client time, which is not significant). Fig. 4(a) shows the results for varying client modulus size k. Note that, as k grows, the time difference between DBProt and KO diminishes. The execution time grows faster than linear with $skey$, due to the overhead of RSA decryption. However, even for strong keys ($skey = 1024$ to 1280 bits), the execution overhead of DBProt is small compared to KO. The trends for varying database size (Fig. 4(b)) confirm that the computational overhead of DBProt is not significant, especially for large database sizes when the overhead of the 1-out-of-2 and blinding steps are negligible compared to the computation of z_i values.



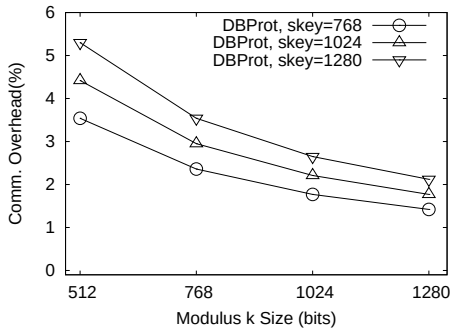
(a) Varying Modulus Size k



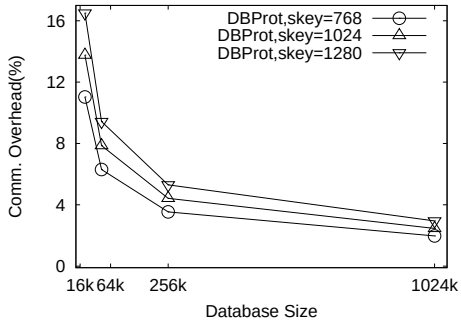
(b) Varying Database Size

Figure 4: Server Execution Time

Fig. 5 shows the communication cost of DBProt, expressed as a percentage of the incurred overhead compared to KO. The overhead decreases when k increases (Fig. 5(a)), as the cost required to transfer the Z array grows faster than the size of the keys transferred in the 1-out-of-2 step. The overhead also decreases as the database grows larger (Fig. 5(b)). Note that, when the database is small, the constant component of the 1-out-of-2 cost results into higher communication overhead. However, for database sizes of 64K elements or more, the overhead never exceeds 10%.



(a) Varying Modulus Size k



(b) Varying Database Size

Figure 5: Communication Overhead

7. CONCLUSIONS

In this paper, we identified vulnerabilities common to the most prominent existing computational PIR protocols which can be exploited by an adversary in order to gain knowledge on large amounts of data stored at a database server. We also proposed extensions that protect the data stored at the server, in addition to the privacy of the client.

In future work, we plan to evaluate the data disclosure of block-PIR protocols [GR05], for which the potential for excessive disclosure is even higher. We also plan to investigate PIR protocols based on novel intractability assumptions, and evaluate their vulnerabilities with respect to database protection.

8. REFERENCES

[BIKJF02] Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and Jean-Fran. Breaking the barrier for information-theoretic private information retrieval. In *IEEE Symposium on Foundations of Computer Science*, pages 261–270, 2002.

[BS96] Eric Bach and Jeffrey Shallit. *Algorithmic Number Theory*. MIT Press, Cambridge, MA, USA, 1996.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *IEEE Symposium on Foundations of Computer Science*, pages 41–50, 1995.

[Cha04] Yan-Cheng Chan. Single database private information retrieval with logarithmic communication. In *Proc. of Australasian Conference on Information Security and Privacy (ACISP)*, pages 50–61, 2004.

[CMO00] Giovanni Di Crescenzo, Tal Malkin, and Rafail Ostrovsky. Single database private information retrieval implies oblivious transfer. In *Proc. of EUROCRYPT*, 2000.

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *EUROCRYPT*, pages 402–414. Springer, 1999.

[Fla88] Daniel E. Flath. *Introduction to Number Theory*. John Wiley & Sons, 1988.

[GIKM98] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. In *ACM Symposium on Theory of Computing*, pages 151–160, 1998.

[GKK+08] Gabriel Ghinita, Panos Kalnis, Ali Khoshgozaran, Cyrus Shahabi, and Kian-Lee Tan. Private queries in location based services: anonymizers are not necessary. In *Proceedings of ACM SIGMOD*, pages 121–132, New York, NY, USA, 2008. ACM.

[GR05] Craig Gentry and Zulfikar Ramzan. Single-Database Private Information Retrieval with Constant Communication Rate. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, pages 803–815, 2005.

[IL89] Russell Impagliazzo and Michael Luby. One-way functions are essential for complexity based cryptography. In *FOCS*, pages 230–235, 1989.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: Single database, computationally-private information retrieval. In *IEEE Symposium on Foundations of Computer Science*, pages 364–373, 1997.

[KO00] Eyal Kushilevitz and Rafail Ostrovsky. One-way trapdoor permutations are sufficient for non-trivial single-server private information retrieval. In *Proc. of EUROCRYPT*, 2000.

[Lip05] Helger Lipmaa. An oblivious transfer protocol with log-squared total communication. In *Proc. of Information Security Conference (ISC)*, pages 314–328, 2005.

[MS00] Sanjeev Kumar Mishra and Palash Sarkar. Symmetrically private information retrieval. In *Proc. of INDOCRYPT 2000*, pages 225–236, 2000.

[NP99] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 245–254, 1999.

[NR06] Krishna Suri Narayanam and C. Pandu Rangan. A Novel Scheme for Single Database Symmetric Private Information Retrieval. In *Proceedings of Annual Inter Research Institute Student Seminar in Computer Science (IRISS)*, pages 803–815, 2006.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[Rab81] Michae O. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Aiken Computation Laboratory, Harvard University, 1981.

[Yek07] Sergey Yekhanin. *Locally decodable codes and private information retrieval schemes*. PhD thesis, MIT, 2007.