Improving the Robustness of Private Information Retrieval

Ian Goldberg
David R. Cheriton School of Computer Science
University of Waterloo
200 University Ave. West
Waterloo, ON, Canada N2L 3G1
iang@cs.uwaterloo.ca

Abstract

Since 1995, much work has been done creating protocols for private information retrieval (PIR). Many variants of the basic PIR model have been proposed, including such modifications as computational vs. information-theoretic privacy protection, correctness in the face of servers that fail to respond or that respond incorrectly, and protection of sensitive data against the database servers themselves.

In this paper, we improve on the robustness of PIR in a number of ways. First, we present a Byzantine-robust PIR protocol which provides information-theoretic privacy protection against coalitions of up to all but one of the responding servers, improving the previous result by a factor of 3. In addition, our protocol allows for more of the responding servers to return incorrect information while still enabling the user to compute the correct result.

We then extend our protocol so that queries have information-theoretic protection if a limited number of servers collude, as before, but still retain computational protection if they all collude. We also extend the protocol to provide information-theoretic protection to the contents of the database against collusions of limited numbers of the database servers, at no additional communication cost or increase in the number of servers. All of our protocols retrieve a block of data with communication cost only $O(\ell)$ times the size of the block, where ℓ is the number of servers.

Finally, we discuss our implementation of these protocols, and measure their performance in order to determine their practicality.

1. Introduction

Private information retrieval (PIR) [4] is the task of fetching an item from a database server without the server learning which item you are interested in. In the context of PIR, an "item" is often thought of as a single bit out of an n-bit database, but it could also be a "block" of size b bits. In the latter case, the n-bit database is considered to be composed of n/b blocks, each of size b bits. A number of applications have been proposed for PIR, including patent and pharmaceutical databases [1], online census information [17], and real-time stock quotes [17]. The Pynchon Gate [11] shows how to use PIR for an arguably more realistic purpose: retrieving pseudonymously addressed email; it argues that PIR is a more suitable primitive for this application than previous proposals.

A trivial solution to the PIR problem is simply to ask the server for the whole database and look up the desired bit or block yourself. To make things more interesting (not to mention practical), we analyze the *communication cost* of the protocol—the total number of bits transmitted—and insist that it be *sublinear*; that is, less than n.

There are two main types of PIR: information-theoretic and computational. In information-theoretic PIR, the server is unable to determine any information about your query even with unbounded computing power. In computational PIR (CPIR) [3, 8], the privacy of the query need only be guaranteed against servers restricted to polynomial-time computations. Note that in the information-theoretic case the unbounded power is only to be used to try to compromise your privacy; in either case we still insist that you and the servers use only polynomial-time computations in order to *perform* the protocol.

It is an unsurprising fact that information-theoretic sublinear PIR is impossible with a single server. However, it *is* possible when there are ℓ servers, each with a copy of the database—assuming that the servers do not collude in order to determine your query. A **t-private ℓ-server PIR** is a PIR system in which the privacy of the query is information-theoretically protected, even if up to t of the ℓ servers collude. (Of course, it must be the case that $t < \ell$.)

Beimel and Stahl [2] investigate the case where servers can fail to respond. In this event, it is important that the

)07 IEEE Symposium on Security and Privacy(SP'07)
7695-2848-1/07 $20.00 © 2007 IEEE

IEEE COMPUTEI SOCIETY

client still be able to retrieve her answer. If only k of the ℓ servers need to respond, and no coalition of up to t servers can learn any information about the query, they call such a system **t-private k-out-of-ℓ PIR**. In addition, they examine systems where, of the k servers that replied (out of ℓ total), v of those k are *Byzantine*; that is, they can return incorrect answers, possibly chosen maliciously or possibly computed in error (because, for example, the server may have an out-of-date copy of the database). However, even with these incorrect answers, the client should still be able to reconstruct the correct database item, and as a side effect, determine which of the servers gave incorrect answers. They term this **t-private v-Byzantine-robust k-out-of-ℓ PIR**, and show that such systems exist for $v \le t < \frac{k}{3}$. Yang et al. [17] propose a PIR protocol for which $v \le t < \frac{k}{2}$, but the client's reconstruction of the correct data block in that protocol does not run in polynomial time.

Gertner, Goldwasser, and Malkin [5] consider that keeping ℓ replicated copies of the database may itself be a security or a privacy risk. They examine PIR protocols where no coalition of up to τ *servers* can determine the contents of the database (information-theoretically). They call this **τ-independent PIR**. They show that they can add τ-independence to any PIR protocol at the expense of increasing the number of servers and the communication cost.

In this paper, we improve the robustness of PIR in a number of ways. First, we allow more servers to collude without compromising privacy, while also allowing more servers to be Byzantine. In particular, we construct a t-private v-Byzantine-robust k-out-of-ℓ PIR protocol for any $0 < t < k$ and $v < k - \lfloor\sqrt{kt}\rfloor$. We show this is always a strict improvement over the previous result, except when $(t, k) = (1, 4)$, where it is the same.

Second, we extend this first protocol to handle the case in which more than t servers collude. In existing t-private PIR systems, a coalition of more than t servers can easily reconstruct the query. We produce a PIR system which has *hybrid* privacy protection: if up to t servers collude, the query is protected information-theoretically, as before; however, if more than t servers collude, the query is still protected computationally. This means that coalitions of up to t servers with unbounded computational power, or of up to all ℓ servers with polynomially bounded computational power, will be unable to determine the client's query.

Finally, we give a second extension that can add τ-independence to our protocol, for $0 \le \tau < k - t - v(2 - \frac{v}{k})$, with *no* increase in the number of servers or in communication cost.

Each variant of our protocol has communication cost only $O(\ell)$ times the size of the data block being retrieved.

At the end of this paper, we briefly discuss our implementation of this protocol, and give some performance measurements.

2. Preliminaries

2.1. Notation

We will denote by $\mathbb{Z}_m$ the ring of integers modulo m, and by $\mathbb{Z}_m^*$ the multiplicative group of invertible integers modulo m. For primes p, we will denote by $\mathbb{F}_p$ the finite field of integers modulo p.

We will denote by δ_{ij} the Kronecker delta function; that is: $\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \ne j \end{cases}$.

Let ϵ be the empty string, and $s||t$ be the concatenation of strings s and t.

2.2. Shamir secret sharing

Sharing of finite field elements. Our technique is based on Shamir secret sharing [14], which we will briefly review. Given a finite field $\mathbb{F}$, and a secret $\sigma \in \mathbb{F}$, we can construct *t-private ℓ-way shares* of the secret in the following way:

1. Choose ℓ distinct non-zero elements $\alpha_1, \ldots, \alpha_\ell$ of $\mathbb{F}$. They can be chosen from any distribution; they need not be uniformly distributed. It is even acceptable to simply use $\alpha_1 = 1, \alpha_2 = 2$, etc. when $\{1, 2, \ldots, \ell\} \subseteq \mathbb{F}\backslash\{0\}$. We call the α_i *indices*.

2. Select t elements $\sigma_1, \ldots, \sigma_t$ of $\mathbb{F}$ uniformly at random.

3. Construct the polynomial $f(x) = \sigma + \sigma_1 x + \sigma_2 x^2 + \cdots + \sigma_t x^t$.

4. The ℓ shares are $f(\alpha_1), \ldots, f(\alpha_\ell)$.

Given any $t + 1$ of the shares, one can recover the polynomial f by Lagrange interpolation and thus determine $\sigma = f(0)$. However, given only t or fewer shares, no information at all about σ is revealed. Because of this, dividing a secret into t-private ℓ-way shares in this way is also called *$(t + 1)$-of-ℓ Shamir secret sharing*.

Sharing of ring elements. Common choices for $\mathbb{F}$, above, include $GF(2^d)$ and $\mathbb{F}_p$. But with minor care, it turns out that the above technique works in some non-fields as well. The proof of the technique from [14] only requires that we are in a finite commutative ring, that α_i is invertible for each $1 \le i \le \ell$ and that $\alpha_i - \alpha_j$ is invertible for each $1 \le i < j \le \ell$.

In particular, we will wish to share elements of rings $\mathbb{Z}_{pq}$ of integers modulo products of two distinct primes p and q. Note that p and q do not need to be a secret. In this scenario, shares are constructed and the secret is reconstructed in exactly the same way as before. The only caveat is in the selection of the α_i. Whereas in the case of a finite field, we only needed that the α_i be non-zero and

)07 IEEE Symposium on Security and Privacy(SP'07)
7695-2848-1/07 $20.00 © 2007 IEEE

IEEE COMPUTEI SOCIETY

distinct, in the modulo pq case, we need that the α_i be non-zero and distinct modulo each of p and q separately. An easy way to ensure this is to choose the α_i from the set $\{1, 2, \ldots, \min(p, q) - 1\}$.

Sharing of vectors. Let $\vec{v}$ be a vector $[v_1, \ldots, v_r]$ of length r, whose entries are elements of either a finite field or a ring $\mathbb{Z}_{pq}$, as above. We can make t-private ℓ-way shares of $\vec{v}$ by simply independently sharing each of the entries. That is, if $x_{j1}, \ldots, x_{j\ell}$ are t-private ℓ-way shares of v_j (for $1 \leq j \leq r$), then $[x_{11}, \ldots, x_{r1}], \ldots, [x_{1\ell}, \ldots, x_{r\ell}]$ are t-private ℓ-way shares of $\vec{v}$.

2.3. The Paillier cryptosystem

The Paillier public-key cryptosystem [10] is another tool we will use. The cryptosystem is as follows:

Key Generation: Select random primes p and q of some desired length, and set $m = pq$ and $\lambda = \text{lcm}(p-1, q-1)$. Define the function $L(u) = (u-1)/m$. Choose a random $g \in \mathbb{Z}^*_{m^2}$ and ensure that $\mu = (L(g^\lambda \bmod m^2))^{-1} \bmod m$ exists. The public encryption key is then (m, g) and the private decryption key is (λ, μ).

Encryption: To encrypt a plaintext $P \in \mathbb{Z}_m$, select a random $\rho \in \mathbb{Z}^*_m$, and compute the ciphertext to be $C = \mathcal{E}(P) = g^P \cdot \rho^m \bmod m^2$. Note that, as usual, $\mathcal{E}$ is a randomized function.

Decryption: To decrypt a ciphertext C, compute $\mathcal{D}(C) = L(C^\lambda \bmod m^2) \cdot \mu \bmod m$.

Note that it is of course the case that $\mathcal{D}(\mathcal{E}(P)) = P$ for all $P \in \mathbb{Z}_m$.

The security of the Paillier cryptosystem is based on the *Decisional Composite Residuosity Assumption* (DCRA). That is, for a fixed public key m, this system is semantically secure if and only if an adversary cannot determine whether or not a given random element of $\mathbb{Z}^*_{m^2}$ has an m^{th} root.

The Paillier cryptosystem has one additional property that is vital for our purposes. It is *additive homomorphic*; that is, multiplying two encryptions together (modulo m^2) gives an encryption of the *sum* of the original messages (modulo m). Formally, $\mathcal{D}(\mathcal{E}(P_1) \cdot \mathcal{E}(P_2) \bmod m^2) = P_1 + P_2 \bmod m$.

3. Improving Byzantine robustness

We motivate our study of Byzantine robustness by looking at the Pynchon Gate [11]. The Pynchon Gate is a system that uses private information retrieval to enable the delivery of email to pseudonymous recipients. Greatly simplified, the system works like this:

- Email arrives at the mail server, destined for a pseudonymous user, say `<wiseone@pynchon.example>`.
- The mail server encrypts the message using a key known by the owner of the pseudonym, and puts the encrypted message in the PIR database (distributing it to ℓ database servers). Note that the server does not know *who* the owner of the pseudonym is.
- At some point, Joe (the owner of the pseudonym) does a PIR query on the database to retrieve the mail for the pseudonym `<wiseone@pynchon.example>`. The privacy guarantees of the PIR technique assure that, unless all ℓ database servers collude, they will be unable to link the client of the query, Joe, to the value of the query, `<wiseone@pynchon.example>`.
- Joe decrypts and reads the resulting message.

The Pynchon Gate uses a PIR protocol from Chor et al. [4], which is shown in Figure 1. It is straightforward to see that this is an $(\ell-1)$-private ℓ-server PIR with information-theoretic protection.[1] Its communication cost is $\ell(r + b) = \ell(n/b + b)$. Choosing b to be $\sqrt{n}$ gives a cost of $2\ell\sqrt{n}$.

However, as reported by Sassaman and Preneel [12], this protocol has a weakness in the presence of Byzantine servers: Joe will be unable to reconstruct the message. Worse, although the Pynchon Gate guarantees Joe will be able to tell that *some* server was Byzantine, he will be unable to tell *which* server it was. Therefore, it is important to produce PIR protocols that not only can allow the client to reconstruct the correct answer, but will also let the client know which servers were Byzantine.

To accomplish this goal, we note that steps P2 and P3 of the Pynchon Gate PIR protocol in Figure 1 form $(\ell-1)$-private ℓ-way shares of the secret e_β (though not with Shamir's method). We replace these steps with a more general t-private ℓ-way Shamir secret sharing of e_β. Note that bitstrings of length r are equivalent to vectors of length r over $\mathbb{F}_2$. We now consider e_β not as a vector over $\mathbb{F}_2$, but rather as a vector of length r over some larger structure $\mathbb{S}$. We still have that the β^{th} entry of e_β is 1, and the other entries are 0, but now these entries are elements of $\mathbb{S}$, and not just $\mathbb{F}_2$. $\mathbb{S}$ might be a field (such as $GF(2^d)$ for some d, or $\mathbb{F}_p$ for some prime p) or a ring $\mathbb{Z}_{pq}$ for some distinct primes p and q. Let $\mathbb{I}$ be a set of *Shamir indices* in $\mathbb{S}$; that is, if $\mathbb{S}$ is a field, $\mathbb{I}$ can just be the non-zero elements of $\mathbb{S}$; if $\mathbb{S}$ is $\mathbb{Z}_{pq}$, $\mathbb{I}$ can be the set $\{1, 2, \ldots, \min(p, q) - 1\}$, as in section

[1] The authors of the Pynchon Gate [11] mistakenly claim that, as an optimization, the client may send $\ell - 1$ of the servers a key for a stream cipher instead of a randomly generated bit string of length r. In reality, doing so reduces the protection provided from information-theoretic to computational.

```
Parameters:
    $\ell$: number of servers
    $n$: size of the database (in bits)
    $b$: size of each block (in bits)
Calculate:
    $r$: number of blocks $= n/b$

Client (querying for block number $\beta$):
    P1. Let $e_\beta$ be the bit string of length $r$ that is all 0s, except for position $\beta$, which is 1.
    P2. Generate $\ell - 1$ random bit strings $\rho_1, \ldots, \rho_{\ell-1}$, each of length $r$.
    P3. Compute $\rho_\ell = \rho_1 \oplus \cdots \oplus \rho_{\ell-1} \oplus e_\beta$.
    P4. Send $\rho_i$ to server number $i$, for $1 \leq i \leq \ell$.

Each server:
    S1. Receive $\rho_i = \rho_{i1} \cdots \rho_{ir}$, a bitstring of length $r$.
    S2. Let $B_j$ be the $j^{\text{th}}$ $b$-bit block of the database for $1 \leq j \leq r$.
    S3. Compute $R_i$ to be the XOR of all the $B_j$ for which $\rho_{ij} = 1$.
    S4. Return $R_i$ to the client.

Client:
    C1. Receive $R_1, \ldots, R_\ell$.
    C2. Compute $B = R_1 \oplus \cdots \oplus R_\ell$.
```

Figure 1. The PIR protocol from Chor et al. [4] used by the Pynchon Gate [11].

2.2. The only restriction on $\mathbb{S}$ is that $\mathbb{I}$ have at least ℓ elements, though we will see later that it sometimes benefits us to choose substantially larger $\mathbb{I}$. We take ℓ random elements of $\mathbb{I}$ as the indices in the Shamir secret sharing, and use them to produce the ρ_i. An important property of Shamir indices is that our usual intuitions about polynomials over fields, such as distinct degree t polynomials agreeing on at most t points, continue to hold in a ring setting provided we restrict our attention to indices selected from $\mathbb{I}$.

Similarly to e_β, in the Pynchon Gate protocol we can consider the ρ_i to be vectors of length r over $\mathbb{F}_2$, and the R_i and B_j to be vectors of length b over $\mathbb{F}_2$. In this case, we see that the computation of R_i in step S3 is the same as computing $R_{ic} = \sum \rho_{ij} B_{jc}$ over $\mathbb{F}_2$ for $1 \leq c \leq b$. When we move to a larger structure $\mathbb{S}$, the servers perform this same computation, but over $\mathbb{S}$.

The secret recovery is more complicated than that of the Pynchon Gate, not only since recovering a secret from Shamir shares is more complicated than recovering from a simple $(\ell-1)$-private ℓ-way XOR scheme, but also since we will need to handle Byzantine servers. The recovery scheme will use the following function Γ: Given a structure $\mathbb{S}$, a list of ℓ indices $[\alpha_1, \ldots, \alpha_\ell]$ in $\mathbb{S}$, a list of ℓ values $[R_1, \ldots, R_\ell]$ in $\mathbb{S} \cup \{\perp\}$, and a polynomial ϕ over $\mathbb{S}$, let $\Gamma(\phi)$ be the subset of $\{1, \ldots, \ell\}$ such that $\phi(\alpha_i) = R_i$ for $i \in \Gamma(\phi)$. Note that we keep $\mathbb{S}$ and the lists α_i and R_i implicit in the notation for convenience.

The resulting PIR protocol is shown in Figure 2. It uses two subroutines, EasyRecover and HardRecover, which are shown in Figure 3. An important fact about these subroutines is that EasyRecover is just a less computationally expensive method to get the same answer as HardRecover, but it only works some of the time:

Fact 1. *If* EasyRecover *returns a non-empty set on a given input, then* HardRecover *will return the same set on that same input.*

Proof. HardRecover finds all polynomials ϕ of degree at most t for which $|\Gamma(\phi)| \geq h$, where $h = k - v$ is the desired minimum number of *honest* (non-Byzantine) servers. EasyRecover is a less expensive procedure to perform the same calculation, in the event that there is exactly one such polynomial. EasyRecover selects $t+1$ of the servers at random, and optimistically assumes that all of those servers returned the correct answer. It calculates the ϕ uniquely determined by those servers' answers, and sees how many other servers gave answers consistent with that polynomial. The key is that if fewer than $h - t$ servers *disagreed*, then there can be no other polynomial ϕ' for which $|\Gamma(\phi')| \geq h$: ϕ' would have to agree with ϕ on more than t points of $\mathbb{I}$, and so $\phi' = \phi$. □

007 IEEE Symposium on Security and Privacy(SP'07)
7695-2848-1/07 $20.00 © 2007 IEEE

```
Parameters:
    $\ell$: number of servers
    $t$: the desired privacy level; that is, the number of servers that can collude without learning
            anything about the query
    $n$: size of the database (in bits)
    $b$: size of each block (in bits)
    $w$: size of each word within a block (in bits)
    $\mathbb{S}$: either a field or a ring $\mathbb{Z}_{pq}$ such that $|\mathbb{S}| \geq 2^w$ (so that each word can be represented by an element of $\mathbb{S}$)
    $\mathbb{I}$: a set of Shamir indices from $\mathbb{S}$ such that $|\mathbb{I}| \geq \ell$
Calculate:
    $r$: number of blocks $= n/b$
    $s$: number of words per block $= b/w$

Client (querying for block number $\beta$):
    P1. Choose $\ell$ random distinct indices $\alpha_1, \ldots, \alpha_\ell$ from $\mathbb{I}$.
    P2. Choose $r$ random polynomials $f_1, \ldots, f_r$ of degree $t$. The coefficients of each
            polynomial should be random elements of $\mathbb{S}$, except for the constant terms.
            The constant term of $f_j$ should be $\delta_{j\beta}$.
    P3. Compute $\rho_i = [f_1(\alpha_i), \ldots, f_r(\alpha_i)]$ for $1 \leq i \leq \ell$.
    P4. Send $\rho_i$ to server number $i$, for $1 \leq i \leq \ell$.

Each (honest) server:
    S1. Receive $\rho_i = [\rho_{i1}, \ldots, \rho_{ir}]$, a vector of $r$ elements of $\mathbb{S}$.
    S2. Let $W_{jc}$ be the $c^{\text{th}}$ $w$-bit word of the $j^{\text{th}}$ $b$-bit block of the database, interpreted as a member of $\mathbb{S}$.
    S3. Compute $R_{ic} = \sum_{1 \leq j \leq r} \rho_{ij} W_{jc}$ for $1 \leq c \leq s$.
    S4. Return $[R_{i1}, \ldots, R_{is}]$ to the client.

Client:
    C1. Receive $[R_{11}, \ldots, R_{1s}], \ldots, [R_{\ell 1}, \ldots, R_{\ell s}]$ from the $\ell$ servers.
            If server $j$ does not respond at all, set $R_{jc} = \perp$ for each $1 \leq c \leq s$.
            Let $\gamma_1, \ldots, \gamma_k$ be the numbers of the $k$ servers which did respond.
            Let $G = \{\gamma_1, \ldots, \gamma_k\}$ and $H = \{(G, \epsilon)\}$.
    C2. If $k \leq t$, abort with the error "not enough servers replied".
    C3. Select $h$ (the minimum number of honest servers) from the range $\sqrt{kt} < h \leq k$.
    C4. For $c$ from 1 to $s$:
    C5.         Set $H' \leftarrow \textsc{EasyRecover}(\mathbb{S}, w, t, h, H, [R_{1c}, \ldots, R_{\ell c}], [\alpha_1, \ldots, \alpha_\ell])$
    C6.         If $H'$ is the empty set, set $H' \leftarrow \textsc{HardRecover}(\mathbb{S}, w, t, h, H, [R_{1c}, \ldots, R_{\ell c}], [\alpha_1, \ldots, \alpha_\ell])$
    C7.         If $H'$ is the empty set, abort with the error "not enough honest servers replied".
    C8.         Set $H \leftarrow H'$.
    C9. The resulting $H$ will be a non-empty set of pairs $(G, B)$. One of the $B$s will be the correct block;
            see section 3.4 for ways to ensure there is only one such $B$.
```

Figure 2. A t-private v-Byzantine-robust k-out-of-ℓ information-theoretic PIR scheme for $0 < t < k$ and $v < k - \lfloor\sqrt{kt}\rfloor$.

)07 IEEE Symposium on Security and Privacy(SP'07)
7695-2848-1/07 $20.00 © 2007 IEEE

```
Inputs:
        $\mathbb{S}$: the structure used for Shamir secret sharing
        $w$: the number of bits per word
        $t$: the desired privacy level of the PIR protocol
        $h$: the minimum number of honest servers that need to respond ($h > t$)
        $H$: a nonempty set of pairs $(G, \sigma)$ where $G$ is a set of at least $h$ server numbers,
                and $\sigma$ is the portion of the requested block recovered so far, assuming that
                the servers in $G$ were the honest ones. Each $\sigma$ will have the same length.
                No two of the $G$ will have more than $t$ elements in common.
        $[R_1, \dots, R_\ell]$: $t$-private $\ell$-way purported shares of a $w$-bit word that had been encoded as a member
                of $\mathbb{S}$. It must not be the case that $R_j = \perp$ for any $j$ in any of the $G$ in $H$.
        $[\alpha_1, \dots, \alpha_\ell]$: the indices used for the secret sharing
Output:
        Either: (1) a set $H'$ of the same form as $H$, above, but with each $\sigma$ being $w$ bits longer
                than those in the input, or (2) the empty set

EASYRECOVER($\mathbb{S}, w, t, h, H, [R_1, \dots, R_\ell], [\alpha_1, \dots, \alpha_\ell]$):
        E1. Set $H' \leftarrow \{\}$.
        E2. For each $(G, \sigma) \in H$:
                /* Optimistically hope the rest of the servers are honest */
        E3.     Select a random subset $I \subseteq G$ of size $t + 1$.
        E4.     Use Lagrange interpolation to find the unique polynomial $\phi$ over $\mathbb{S}$ of degree $t$
                for which $\phi(\alpha_j) = R_j$ for each $j \in I$.
        E5.     Let $W$ be the $w$-bit representation of $\phi(0)$, or $\perp$ if there is no such representation.
        E6.     If $|G \cap \Gamma(\phi)| \geq h$ and $|G \backslash \Gamma(\phi)| < h - t$ and $W \neq \perp$ then add $(G \cap \Gamma(\phi), \sigma || W)$ to $H'$.
        E7.     Otherwise, immediately return the empty set.
        E8. Return $H'$.

HARDRECOVER($\mathbb{S}, w, t, h, H, [R_1, \dots, R_\ell], [\alpha_1, \dots, \alpha_\ell]$):
        H1. Set $H' \leftarrow \{\}$.
        H2. Use the algorithm of [7] to recover (in polynomial time) the set $\{\phi_i\}$ of polynomials over
                $\mathbb{S}$ of degree $\leq t$ for which $\phi_i(\alpha_j) = R_j$ for at least $h$ values of $j \in \{1, \dots, \ell\}$.
        H3. For each such $\phi_i$:
        H4.     Let $W_i$ be the $w$-bit representation of $\phi_i(0)$, or $\perp$ if there is no such representation.
        H5.     If $W_i \neq \perp$, then for any $(G, \sigma) \in H$ such that $|\Gamma(\phi_i) \cap G| \geq h$,
                add $(\Gamma(\phi_i) \cap G, \sigma || W_i)$ to $H'$.
        H6. Return $H'$.
```

Figure 3. The EASYRECOVER and HARDRECOVER subroutines.

)07 IEEE Symposium on Security and Privacy(SP'07)
7695-2848-1/07 $20.00 © 2007 IEEE

Note that in the presence of Byzantine servers, EASYRECOVER may not always find the unique polynomial, even if there is one, but in no case will it output a non-empty set when more possibilities exist.

3.1. Privacy of the protocol

It is easy to see that no coalition of up to t servers can learn any information about β (the requested block number): between them, they have at most t of the t-private ℓ-way shares of the vector e_β (as in section 2.2). By the properties of Shamir secret sharing, they learn no information about e_β, and therefore about β.

It is important to note that this result holds even if some servers are Byzantine, since all of the information flowing from the client to the servers happens *before* the servers perform any actions.

3.2. Correctness of the protocol without Byzantine servers

In this section, we show that this protocol returns the correct block B from the database when no server responds incorrectly (but some may not respond at all), so long as enough servers do respond.

Theorem 1. *If k of the ℓ servers respond, there are no Byzantine servers, and $k > t$, then the algorithm in Figure 2 will return a set H containing the single pair (G, B), where G is the set of the k server numbers that responded, and B is the correct block B_β from the database.*

Proof sketch. (See Appendix A for the complete proof.) The important observation is that if ℓ vectors $\rho_1, \ldots, \rho_\ell$ are t-private ℓ-way secret shares of a vector $\vec{v}$ of length r, and $\vec{w}$ is any vector of length r, then the ℓ dot products $\rho_1 \cdot \vec{w}, \ldots, \rho_\ell \cdot \vec{w}$ are t-private ℓ-way secret shares of the scalar $\vec{v} \cdot \vec{w}$.

In this protocol, each server has its share ρ_i of $\vec{v} = e_\beta$. Also, for each c from 1 to s (where s is the number of words per block), each server constructs the vector $\vec{w_c}$, which is the vector of length r (the number of blocks) whose j^{th} element is the c^{th} word of the j^{th} block of the database. Then each server's returned value $R_{ic} = \rho_i \cdot \vec{w_c}$ will be its share of $e_\beta \cdot \vec{w_c}$, which is just the c^{th} word of the β^{th} block of the database. Since the client receives more than t of these results, it can uniquely reconstruct each of the words of the β^{th} block of the database, and concatenating them reproduces the desired block. □

3.3. Correctness in the presence of Byzantine servers

We will now look at the effect of Byzantine servers on the correctness of this protocol.

Theorem 2. *If k of the ℓ servers respond at all, $k > t$, and at least $h > \sqrt{kt}$ servers respond honestly, then the algorithm in Figure 2 will return a set H, one of whose elements is the pair (G_h, B_β), where G_h is the set of the server numbers that responded honestly, and B_β is the correct block from the database.*

Proof sketch. (See Appendix A for the complete proof.) As above, the h honest servers return their shares of the s words of the β^{th} block of the database. However, $k - h$ additional servers return arbitrary values. Depending on the values of h, k and t, there may no longer be a *unique* block determined by h of the k received shares, but when $h > \sqrt{kt}$ we can use the algorithm of [7] to find a list of *all* possible blocks in polynomial time. In the next section, we will see a number of ways to recover the correct block from this list. □

3.4. List-decoding

The algorithm of Figure 2 is an example of a *list-decoding* algorithm; that is, under some circumstances, it may output a list of *more than one* data block, and we must provide some way for the client to determine which is the correct block. On the one hand, the potential to list-decode is one source of the improvements in the privacy and robustness parameters of the protocol of this section over that of previous work such as Beimel and Stahl [2]. On the other hand, we need to be able to recover the correct database block.

The source of the list-decoding is that, in the presence of many Byzantine servers, there may be more than one polynomial ϕ such that $|\Gamma(\phi)| \geq h$. (Recall that $\Gamma(\phi)$ is the set of server numbers which returned replies consistent with the polynomial ϕ.) There are a number of ways to handle this and recover the unique correct result.

The simplest way to handle list-decoding is simply to choose your parameters such that there cannot be more than one such ϕ.

Fact 2. *If $h > \frac{k+t}{2}$, then there is exactly one polynomial ϕ of degree at most t for which $|\Gamma(\phi)| \geq h$.*

Proof. Suppose the correct polynomial is ϕ_0, so that $|\Gamma(\phi_0)| \geq h$, since all h honest servers will respond correctly. Now suppose it is the case that $|\Gamma(\phi)| \geq h$ for some ϕ. It is always the case that $|\Gamma(\phi_0) \cap \Gamma(\phi)| = |\Gamma(\phi_0)| + |\Gamma(\phi)| - |\Gamma(\phi_0) \cup \Gamma(\phi)|$. But since $|\Gamma(\phi_0) \cup \Gamma(\phi)| \leq k$ (as only k servers responded), we have that $|\Gamma(\phi_0) \cap \Gamma(\phi)| \geq h + h - k > (k + t) - k = t$. So ϕ_0 and ϕ agree on more than t points of $\mathbb{I}$, and are therefore equal. □

Therefore, if $h > \frac{k+t}{2}$, we will never have to handle more than one possible polynomial. However, we may like to be able to use lower values of h. The following fact shows how to handle values of $h > k/2$.

Fact 3. *If the Byzantine servers are unable to see the communication between the client and the honest servers (which should be the case, as it is important for privacy), and $h > k/2$, then by choosing the index set $\mathbb{I}$ to be sufficiently large, we can make the probability that the algorithm of Figure 2 outputs more than one block arbitrarily small.*

Proof sketch. (See Appendix A for the complete proof.) Suppose the algorithm of Figure 2 outputs more than one block. Then it must be the case that the Byzantine servers were able to provide responses such that there is an *incorrect* polynomial ϕ of degree t which agrees with at least h of the k total responses. Since there are at most $k - h$ Byzantine servers, and $k - h < k/2 < h$, we must have that $\phi(\alpha_i) = \phi_0(\alpha_i)$, where ϕ_0 is the correct polynomial, for at least one of the *honest* servers' α_i.

Now the key observation is that the Byzantine servers *do not know* the indices associated with the honest servers. Remember that ϕ can agree with ϕ_0 in at most t places. Therefore, if $\mathbb{I}$, the set of possible indices, is very large, then the probability that one of those places happens to be an index associated with one of the honest servers will be very small. □

Note that if $t \geq \frac{k}{4}$, then we always have $h > \sqrt{kt} \geq k/2$. Also recall that the size of the index set $\mathbb{I}$ can be chosen to be $|\mathbb{S}| - 1$ (if $\mathbb{S}$ is a field) or $\min(p, q) - 1$ (if $\mathbb{S}$ is $\mathbb{Z}_{pq}$).

Finally, in the event that $t < \frac{k}{4}$ and we want to allow for $k/2$ or more Byzantine servers, we can just use the usual techniques (such as those of [9]) to add redundancy to the words of the database, so that the list decoding can be converted to unique decoding. Note that this will slightly increase the size of the database. This redundancy could be in the form of digital signatures from the database creator, for example. In a situation where there is for some reason no such independent creator, allowing a majority of responding servers to be Byzantine may not make sense; in that case, it is not clear what it means for a block to be "correct" when more than half of the servers are storing a different block.

3.5. Comparison to previous results

Privacy and robustness. The authors of [2] show that t-private v-Byzantine-robust k-out-of-ℓ PIR protocols exist for $v \leq t < \frac{k}{3}$. We have demonstrated such a protocol for $0 < t < k$ and $v < k - \lfloor\sqrt{kt}\rfloor$. Our protocol can therefore withstand at least three times as many servers colluding to determine the client's query, and when the privacy level t is the same (for some $0 < t < \frac{k}{3}$), our protocol tolerates up to $k - \lfloor\sqrt{kt}\rfloor - 1$ Byzantine servers, while that of [2] tolerates up to t. We now show that in these comparable cases, our result is always at least as good, and almost always strictly better:

Theorem 3. *For integers k, t such that $0 < t < \frac{k}{3}$, we have $k - \lfloor\sqrt{kt}\rfloor - 1 \geq t$, with equality if and only if $(t, k) = (1, 4)$.*

Proof. We equivalently prove that $\frac{k - \lfloor\sqrt{kt}\rfloor - 1 - t}{t} \geq 0$, with equality if and only if $(t, k) = (1, 4)$.

First note that

$$\begin{aligned} \frac{k - \lfloor\sqrt{kt}\rfloor - 1 - t}{t} &\geq \frac{k - \sqrt{kt} - 1 - t}{t} \\ &= \frac{k}{t} - \sqrt{\frac{k}{t}} - \frac{1}{t} - 1 \\ &= \sqrt{\frac{k}{t}}\left(\sqrt{\frac{k}{t}} - 1\right) - \frac{1}{t} - 1 \end{aligned}$$

with equality if and only if kt is a perfect square. Now $k \geq 3t + 1$, so $\frac{k}{t} \geq 3 + \frac{1}{t}$, and

$$\begin{aligned} &\sqrt{\frac{k}{t}}\left(\sqrt{\frac{k}{t}} - 1\right) - \frac{1}{t} - 1 \\ &\qquad \geq \sqrt{3 + \frac{1}{t}}\left(\sqrt{3 + \frac{1}{t}} - 1\right) - \frac{1}{t} - 1 \\ &\qquad = \left(3 + \frac{1}{t}\right) - \sqrt{3 + \frac{1}{t}} - \frac{1}{t} - 1 \\ &\qquad = 2 - \sqrt{3 + \frac{1}{t}} \end{aligned}$$

with equality if and only if $k = 3t + 1$. Finally, we have that $2 - \sqrt{3 + \frac{1}{t}} \geq 0$, with equality if and only if $t = 1$, and the result is proven. □

Note that $(t, k) = (1, 4)$ is the *minimal* configuration of the system in [2].

Communication cost. This protocol sends $r = n/b$ elements of $\mathbb{S}$ to each of ℓ servers, and receives $s = b/w$ elements of $\mathbb{S}$ from each of k servers in reply. If it takes z bits to encode an arbitrary element of $\mathbb{S}$ (so $z = \lceil\lg(|\mathbb{S}|)\rceil$), then the total communication cost is $n\ell z/b + kbz/w$. Since $k \leq \ell$, this is bounded by $\ell z(n/b + b/w)$. By choosing $b = \sqrt{nw}$, we get $r = s = \sqrt{n/w}$ and the total communication cost to privately retrieve a block of $\sqrt{nw}$ bits is bounded by $2\ell z\sqrt{n/w}$. Remember that we needed to choose $\mathbb{S}$ such that $|\mathbb{S}| \geq 2^w$; if we make it not much bigger, we can have $z = w + 1$, or even $z = w$ if $\mathbb{S}$ is $GF(2^w)$. Then our cost to retrieve $\sqrt{nw}$ bits is $O(\ell\sqrt{nw})$

Note that this is far from the optimal communication cost of retrieving a *single* bit, even in the context of Byzantine-robust PIR protocols; for example, the protocol of [2] has cost $O(\frac{k}{3t} n^{\frac{1}{\lfloor (k-1)/3t \rfloor}} \ell \log \ell)$ to retrieve one bit. However, it is clearly within a small factor of optimal if indeed we are interested in the entire $\sqrt{nw}$-bit block.

4. Robustness against colluding servers

In this section, we consider the problem of more than t (even up to all ℓ) servers colluding to try to determine the client's query.

As mentioned earlier, if all ℓ servers collude, it is impossible to make a protocol with communication cost less than n which also information-theoretically protects the query. Therefore, we do the best possible thing: information-theoretically protect the query if up to t servers collude, but still computationally protect the query even if up to all ℓ servers collude. We call a PIR protocol with this property **t-private ℓ-computationally-private**.

We do this with a simple modification to the protocol of Figure 2: instead of sending t-private ℓ-way shares of e_β to the servers, send *encryptions* of those shares, under an additive homomorphic cryptosystem, such as the Paillier cryptosystem [10] (see section 2.3). The servers then use the homomorphic property to compute the encryptions of their results, which they send back to the client. The client decrypts the replies and proceeds as before.

In detail, the changes to the protocol of Figure 2 are:

- To start, select large random distinct primes p and q. Set $m = pq$, choose $\mathbb{S}$ to be the ring $\mathbb{Z}_m$, and let $\mathbb{I} = \{1, 2, \ldots, \min(p, q) - 1\}$. Compute the Paillier encryption and decryption keys as in section 2.3.
- In step P3, use Paillier encryption to compute $\mathcal{E}(\rho_i) = [\mathcal{E}(f_1(\alpha_i)), \ldots, \mathcal{E}(f_r(\alpha_i))]$.
- In step P4, send $\mathcal{E}(\rho_i)$ to server i, for $1 \leq i \leq \ell$.
- In step S1, $\mathcal{E}(\rho_i) = [\mathcal{E}(\rho_{i1}), \ldots, \mathcal{E}(\rho_{ir})]$ will be a vector of r elements of $\mathbb{Z}_{m^2}$.
- In step S3, compute $\mathcal{E}(R_{ic}) = \prod_{1 \leq j \leq r} \mathcal{E}(\rho_{ij})^{W_{jc}}$ as elements of $\mathbb{Z}_{m^2}$ for $1 \leq c \leq s$.
- In step S4, return $[\mathcal{E}(R_{i1}), \ldots, \mathcal{E}(R_{is})]$ to the client.
- In step C1, use Paillier decryption to compute $R_{ic} = \mathcal{D}(\mathcal{E}(R_{ic}))$, and then proceed as before.

This modified protocol still allows the client to recover the desired block B_β, even when only k of the ℓ servers respond, and $v < k - \lfloor \sqrt{kt} \rfloor$ of those k are Byzantine. This fact follows immediately from the equivalent result for the original protocol in Figure 2: once the client receives and decrypts the servers' replies, he has the same information as he would have had, had none of the encryption or decryption happened.

The information-theoretic protection of the client's query against coalitions of up to t servers is also immediate: if a coalition of t servers, knowing t of the ρ_i, cannot learn any information about β, then certainly if those servers instead know $\mathcal{E}(\rho_i)$, that does not give them *more* information about β. Formally, given any algorithm $\mathcal{A}$ that can recover information about β given t of the $\mathcal{E}(\rho_i)$, one can easily construct an algorithm $\mathcal{A}'$ that recovers that same information about β given t of the ρ_i, by first encrypting the ρ_i, and passing the results to $\mathcal{A}$. Since there is no such $\mathcal{A}'$, there is also no such $\mathcal{A}$.

Now we turn our attention to the case in which up to all ℓ of the servers collude. The privacy consideration is only interesting if there is more than one block in the database, so we assume $r \geq 2$.

Theorem 4. *Given a fixed m, if there is a probabilistic polynomial time algorithm $\mathcal{A}$ which can distinguish Paillier encryptions of t-private ℓ-way Shamir secret shares of the vector e_1 from Paillier encryptions of t-private ℓ-way Shamir secret shares of the vector e_2 with some probability ψ, then there is a probabilistic polynomial time algorithm $\mathcal{A}'$ which can distinguish Paillier encryptions of the number 0 from Paillier encryptions of the number 1 with the same probability ψ.*

Proof. Suppose the given algorithm $\mathcal{A}$ takes as input Paillier encryptions of t-private ℓ-way secret shares of e_β for some $\beta \in \{1, 2\}$, and, to be generous, the ℓ indices used for the Shamir secret sharing, and $\mathcal{A}$ outputs $\beta' \in \{1, 2\}$. Then by assumption, $\Pr[\beta' = 1 | \beta = 1] - \Pr[\beta' = 1 | \beta = 2] = \psi$.

The desired algorithm $\mathcal{A}'$ is as follows:

Input:

A ciphertext $C = \mathcal{E}(\zeta)$, for some $\zeta \in \{0, 1\}$.

Output:

A guess ζ' at the value of ζ.

Algorithm:

1. Choose ℓ random distinct indices $\alpha_1, \ldots, \alpha_\ell$ from $\mathbb{Z}_m$. Verify that $\gcd(\alpha_i, m) = 1$ for each i and that $\gcd(\alpha_i - \alpha_j, m) = 1$ for each i, j.
2. Choose r random polynomials $f_1, \ldots, f_r$ of degree t. The coefficients of each polynomial should be random elements of $\mathbb{Z}_m$, except for the constant terms. The constant term of each f_j should be 0.
3. Compute $\mathcal{E}(\rho_i) = [\mathcal{E}(f_1(\alpha_i)) \cdot C, \mathcal{E}(f_2(\alpha_i)) \cdot \mathcal{E}(1)/C, \mathcal{E}(f_3(\alpha_i)), \ldots, \mathcal{E}(f_r(\alpha_i))]$ for $1 \leq i \leq \ell$.
4. Output $\zeta' =$ $(2 - \mathcal{A}([\mathcal{E}(\rho_1), \ldots, \mathcal{E}(\rho_\ell)], [\alpha_1, \ldots, \alpha_\ell]))$.

To see why this works, notice that $[f_j(\alpha_1), \ldots, f_j(\alpha_\ell)]$ are t-private ℓ-way secret shares of the value 0, for each $1 \leq j \leq r$. Therefore $[f_1(\alpha_1) + \zeta, \ldots, f_1(\alpha_\ell) + \zeta]$ are t-private ℓ-way secret shares of the value ζ, and $[f_2(\alpha_1) + 1 - \zeta, \ldots, f_2(\alpha_\ell) + 1 - \zeta]$ are t-private ℓ-way secret shares of the value $1 - \zeta$. But by the homomorphic property of Paillier encryption, $\mathcal{E}(f_1(\alpha_i)) \cdot C$ is an encryption of $f_1(\alpha_i) + \zeta$, and $\mathcal{E}(f_2(\alpha_i)) \cdot \mathcal{E}(1)/C$ is an encryption of $f_2(\alpha_i) + 1 - \zeta$, so $[\mathcal{E}(\rho_1), \ldots, \mathcal{E}(\rho_\ell)]$ are t-private ℓ-way shares of e_2 if $\zeta = 0$, and of e_1 if $\zeta = 1$. Therefore $\Pr[\zeta' = 1 | \zeta = 1] - \Pr[\zeta' = 1 | \zeta = 0] = \Pr[\beta' = 1 | \beta = 1] - \Pr[\beta' = 1 | \beta = 2] = \psi$, and the result is proven. □

Remembering from section 2.3 that the the Paillier cryptosystem is semantically secure if and only if the Decisional Composite Residuosity Assumption (DCRA) holds (for the value of m used in the cryptosystem), we get the following:

Corollary. *The protocol of this section maintains the privacy of the query against coalitions of up to all ℓ servers, provided the Decisional Composite Residuosity Assumption holds for those servers, for the value of m used in the protocol.*

That is, so long as the servers do not have sufficient computational power to break the DCRA, they will be unable to distinguish queries for block 1 from queries for block 2. By symmetry (there is nothing special about blocks 1 and 2), the servers will be unable to distinguish any one query from another.

4.1. Communication cost

The only change to the communication cost is that elements of $\mathbb{Z}_{m^2}$ are being sent between the client and the servers, instead of elements of $\mathbb{S} = \mathbb{Z}_m$. This causes all communication to approximately double in size, increasing the communication cost by the constant factor of 2; the cost is still $O(\ell\sqrt{nw})$ to retrieve $\sqrt{nw}$ bits.

4.2. Notes on the choices of parameters

It should be noted that in both the original scheme given in Figure 2, and this modification, the choice of almost all the parameters, including t, b, w, $\mathbb{S}$, $\mathbb{I}$, the α_i, and h, is done *by the client*. Furthermore, each client using the same database can choose his own values for the parameters independently of any other clients' choices. (In that case, however, b, w, and $\mathbb{S}$ need to be communicated to the server during the protocol, marginally increasing the communication cost.)

$\mathbb{S}$ should be chosen as small as needed to achieve the desired security properties, since the communication cost of the protocol depends on the size of $\mathbb{S}$. Note, however, that although using a larger $\mathbb{S}$ will result in a higher communication cost, the client will also retrieve a correspondingly larger database block.

For example, in the protocol of this section, the DCRA needs to hold over $\mathbb{S} = \mathbb{Z}_m$, so m needs to be chosen to be at least 1024 bits long (since the DCRA is clearly at most as hard as factoring m).

On the other hand, if the protocol is not required to be ℓ-computationally-private, much smaller structures $\mathbb{S}$ will do. Minimally, we must have $|\mathbb{S}| \geq \ell + 1$, and this value suffices if $h > \frac{k+t}{2}$, as in Fact 2, or if we are using redundancy to avoid list decoding. If we are using Fact 3 to avoid list decoding, then we will probably want to choose $|\mathbb{S}|$ to be around 2^{128}.

Once we have selected $\mathbb{S}$, the best choices for w and b are then $\lfloor \lg(|\mathbb{S}|) \rfloor$ and $\sqrt{nw}$, respectively.

5. Protecting the data from the servers

In this section, we give a small enhancement to the protocols of the previous sections that allows the contents of the database itself to be hidden from coalitions of up to τ servers, for $0 \leq \tau < k - t - v(2 - \frac{v}{k})$. We achieve *$\tau$-independence*, as defined in [5]: no coalition of up to τ servers has any information about the content of the database (in the information-theoretic sense). Unlike the result in [5], however, we do not achieve τ-independence at the expense of an increased number of servers or at the expense of communication cost: the number of servers and communication cost of the τ-independent version of our scheme are identical to those of the regular version.

The major change we make to our protocol in order to achieve τ-independence is that, in this scheme, the choices of $\mathbb{S}$, $\mathbb{I}$, and the indices α_i need to be made *in advance* of storing data in the database. This condition imposes the following restrictions on the use of the scheme:

- If it is intended that the user storing the information in the database is different from the client retrieving the data, or if there is more than one such client, they cannot rely on the secrecy of the α_i to get the benefit of Fact 3. They need to use redundancy techniques instead, as mentioned in section 3.4, or reduce the allowed number of Byzantine servers to at most $\frac{k-t-\tau}{2}$.
- If it is intended that there is more than one client retrieving data, this scheme cannot be used at the same time as the scheme from section 4: in the latter scheme, $\mathbb{S}$ was chosen to be $\mathbb{Z}_{pq}$ for secret values p and q. Multiple clients would not use the same $\mathbb{S}$, and so with this variant, could not use the same database at all.

At system setup time, $\mathbb{S}$ and the α_i are chosen, and communicated to all of the users of the database (either users storing data, or users retrieving data). $\mathbb{S}$ must be communicated to the servers as well, but the α_i need not be.

As before, the database is divided into $r = n/b$ b-bit blocks, and each block is divided into $s = b/w$ w-bit words. But instead of the server i storing the s words $W_{j1}, \ldots, W_{js}$ of block number j directly, it stores each block as a sequence of s elements $\omega_{j1}^{(i)}, \ldots, \omega_{js}^{(i)}$ of $\mathbb{S}$.

The computation of these $\omega_{jc}^{(i)}$ uses Shamir secret sharing. In particular, a user that wants to store database block j divides it into s words $W_{j1}, \ldots, W_{js}$, and does the following for each $1 \leq c \leq s$:

- Choose a random polynomial g_{jc} of degree τ. The coefficients of g_{jc} should be random elements of $\mathbb{S}$, except for the constant term, which should be W_{jc} (encoded as a member of $\mathbb{S}$).

- Send $g_{jc}(\alpha_i)$ to server i to store as its $\omega_{jc}^{(i)}$, for each $1 \leq i \leq \ell$.

That is, the values of $\omega_{jc}^{(i)}$ for $1 \leq i \leq \ell$ are just τ-private ℓ-way Shamir secret shares of W_{jc}.

The modifications to the protocol of Figure 2 are now straightforward:

- Remove step S2, and use $\omega_{jc}^{(i)}$ instead of W_{jc} in step S3.

- In step C2, check that $k \leq t + \tau$ instead of $k \leq t$.

- In step C3, choose h from the range $\sqrt{k(t+\tau)} < h \leq k$ instead of $\sqrt{kt} < h \leq k$.

- In steps C5 and C6, pass $t + \tau$ instead of t.

We note that our choice of $0 \leq \tau < k - t - v(2 - \frac{v}{k})$ ensures that the same values of h, v, t, k, and ℓ we used in the original protocol will continue to work in the τ-independent version. In particular, choosing τ from this range guarantees that $k \leq t + \tau$ and $\sqrt{k(t+\tau)} < h \leq k$.

We also note that if we set $\tau = 0$, we get exactly the same protocol as before, since 0-private ℓ-way shares of W_{jc} are just ℓ copies of W_{jc} itself.

Why does this work? Step S3 computes R_{ic} to be $\sum_{1 \leq j \leq r} \rho_{ij}\omega_{jc}^{(i)} = \sum_{1 \leq j \leq r} f_j(\alpha_i) g_{jc}(\alpha_i) = F_c(\alpha_i)$, where F_c is the polynomial $F_c = \sum_{1 \leq j \leq r} f_j g_{jc}$ of degree at most $t + \tau$. Note, however, that it is *not* necessarily the case that the R_{ic} are $(t+\tau)$-private ℓ-way shares of $F_c(0) = W_{\beta c}$, since the distribution of the F_c is not uniform. In particular, it may be possible to learn some information about $W_{\beta c}$ given $t + \tau$ of the $F_c(\alpha_i)$. However, it *is* still the case that any $t + \tau + 1$ of the $F_c(\alpha_i)$ uniquely determines F_c, and that is the only fact we use in our reconstruction of $W_{\beta c}$.

Therefore, we have constructed a t-private v-Byzantine-robust τ-independent k-out-of-ℓ PIR protocol for $0 < t \leq t + \tau < k$, and $v < k - \lfloor\sqrt{k(t+\tau)}\rfloor$. This protocol has communication cost $O(\ell\sqrt{nw})$ to retrieve $\sqrt{nw}$ bits of the database. If there is to be only one client retrieving data, we can use both the extensions of this section and of section 4 at the same time, and add ℓ-computationally-private to the list of properties, at a cost of a factor of 2 in the communication.

6. Implementation details

We implemented the protocols in this paper in C++, using Victor Shoup's NTL library [15], except for one part of step H2 of the HARDRECOVER subroutine, which is currently performed by the computer algebra system MuPAD [13]. Our implemenation is available as the Percy++ project on SourceForge [6].

We measured the computational performance of these protocols on a Lenovo T60p laptop computer with a 2.16 GHz dual-core Intel CPU running Ubuntu Linux in order to determine their practicality.

We first measured the performance of the protocol of Figure 2; that is, the protocol without the ℓ-computationally-private and τ-independent properties. We used a range of values of n, t, ℓ, and w, and we set $k = \ell$ in all cases.

Figure 4 shows some representative results. In these cases, the were no Byzantine servers, so the HARDRECOVER subroutine was never executed. Figure 4(a) shows the client's processing time, as a function of the database size, for various values of w. In this plot, we set $(t, k) = (12, 20)$. The plot suggests a square-root dependence on the database size, which agrees with an examination of the algorithm. We confirm this by squaring the measurements; the results are shown in Figure 4(b), which indeed produces linear graphs. Figure 4(c) shows the servers' processing time, and again as expected, this is linear in the database size. Finally, Figure 4(d) shows that for fixed k, the client's running time depends linearly on the privacy level t. As a numeric example, for $w = 128$, the client processing takes $44\sqrt{n}$ microseconds, and the server processing takes $9.6n$ nanoseconds.

When we introduce Byzantine servers, the HARDRECOVER subroutine gets executed. As expected, this is noticeably more expensive than the EASYRECOVER subroutine. For $(t, k) = (5, 10)$, for example, it adds a couple of seconds to the client's processing time. For $(t, k) = (10, 20)$, it adds several *minutes*. However, this is not onerous, since it is likely that the mere ability of the client to detect *which* servers are returning incorrect results will deter the servers from doing so. Therefore, we expect to use the HARDRECOVER subroutine only rarely.

Adding τ-independence (the modification to the protocol

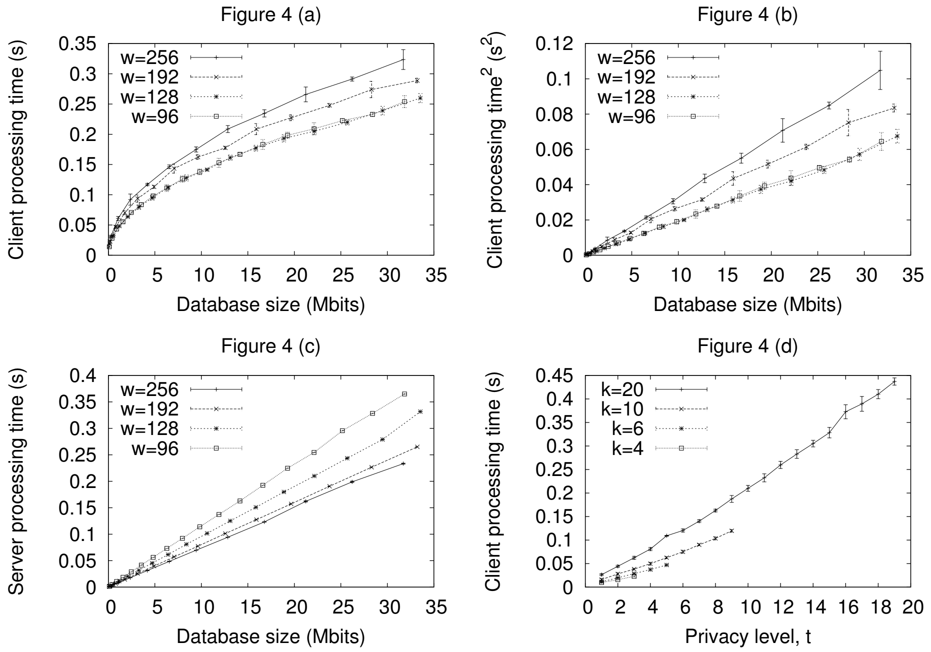

Figure 4. Performance measurements for the protocol of Figure 2.

from Section 5) is, as expected, quite cheap. In Figure 5 we plot timings of the t-private τ-independent version of the protocol. In each graph, we fix $w = 128$, $k = 20$, and $n = 2^{25}$, and vary t and τ such that $0 < t < t + \tau < k$. In Figure 5(a) we see that the server's processing time is independent of both t and τ. We divide the client's processing time into two parts: Figure 5(b) shows the time it takes the client to *prepare* its query (the steps labelled "P" in Figure 2), and Figure 5(c) shows the time it takes the client to *reconstruct* the data block (the steps labelled "C" in Figure 2). The graphs clearly show that the preparation time is linearly dependent on t, but independent of τ, and the reconstruction time is linearly dependent on $t + \tau$, as would be expected from the algorithm. The careful reader will note that the sum of the times in Figures 5(b) and 5(c) is slightly less than the corresponding times in Figure 4(d); this is because the α_i are fixed in the τ-independent version, and are in fact chosen to be the very simple $\alpha_i = i$.

On the other hand, adding ℓ-computational privacy (the modification from Section 4) is quite expensive. The server needs to perform one modular exponentiation for each w-bit word in the database. The plots have the same shape as those of Figure 4, but the scale is different: for $w = 1024$, $k = 5$, and $t = 4$, we find the client's processing time is $15\sqrt{n}$ milliseconds, and the server's processing time is $30n$ microseconds. For values of n in the hundreds of millions of bits or more, these times are substantial.

7. Conclusion

We have improved the robustness of protocols for private information retrieval in a number of ways. Compared to the previous scheme in [2], our basic protocol allows for more servers to collude without compromising the user's privacy. Moreover, maintaining the same privacy level as in [2], we enable the reconstruction of the correct data block when more servers return faulty responses. We extended this protocol to add hybrid privacy protection; that is, information-theoretic protection if up to t servers collude (for some $t < k \leq \ell$), but still computational protection if up to all ℓ collude. Finally, we presented another extension which added τ-independence to the protocol while increasing neither the number of servers, nor the communication cost.

We implemented and measured these protocols and found the performance to agree well with theory. With the exception of the hybrid privacy protection, our implementation gives practical speeds for moderately sized databases.

8. Acknowledgements

We would like to thank Len Sassaman for motivating the problem, and David Molnar, Len Sassaman and Bryce Wilcox-O'Hearn for their helpful discussions and comments. We would also like to thank Katrina Hanna, Bram Cohen, Urs Hengartner, Joel Reardon, Jörn Müller-Quade and the anonymous referees for improving the quality of the paper.

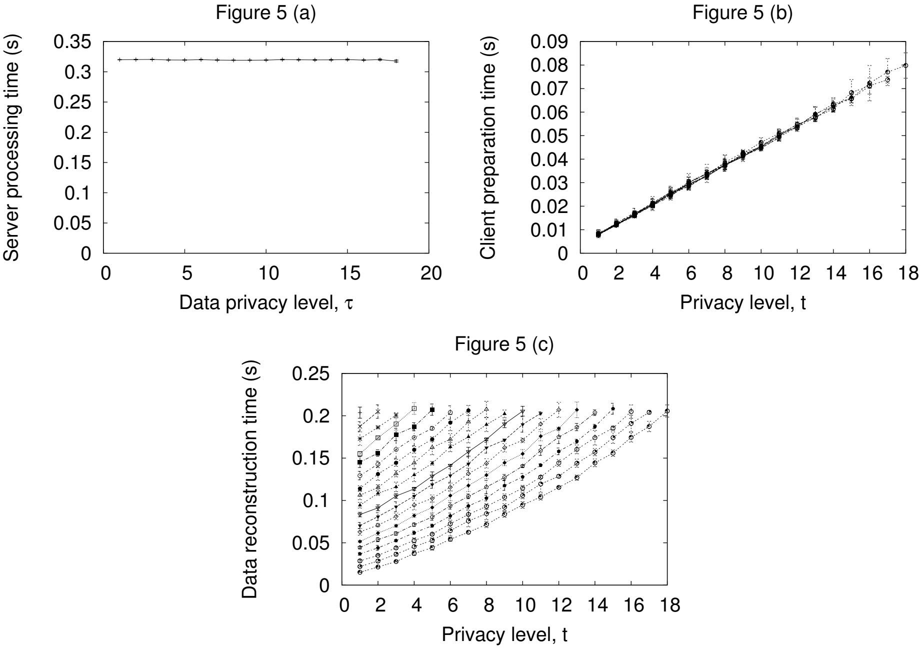

Figure 5. Performance measurements for the τ-independent version of the protocol.

References

[1] D. Asonov. Private Information Retrieval: An overview and current trends. In *Proceedings of the ECDPvA Workshop, Informatik 2001*, September 2001.

[2] A. Beimel and Y. Stahl. Robust Information-Theoretic Private Information Retrieval. In *Third Conference on Security in Communication Networks, Lecture Notes in Computer Science 2576*, pages 326–341. Springer-Verlag, 2002.

[3] B. Chor and N. Gilboa. Computationally Private Information Retrieval. In *29th Annual ACM Symposium on Theory of Computing*, pages 304–313, 1997.

[4] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private Information Retrieval. In *36th Annual Symposium on Foundations of Computer Science*, pages 41–50, 1995.

[5] Y. Gertner, S. Goldwasser, and T. Malkin. A Random Server Model for Private Information Retrieval. In *2nd International Workshop on Randomization and Approximation Techniques in Computer Science, Lecture Notes in Computer Science 1518*, pages 200–217. Springer-Verlag, 1998.

[6] I. Goldberg. Percy++ project on SourceForge. http://percy.sourceforge.net/.

[7] V. Guruswami and M. Sudan. Improved Decoding of Reed-Solomon and Algebraic-Geometry Codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, September 1999.

[8] E. Kushilevitz and R. Ostrovsky. Replication is Not Needed: Single Database, Computationally Private Information Retrieval. In *38th Annual Symposium on Foundations of Computer Science*, pages 364–373, October 1997.

[9] S. Micali, C. Peikert, M. Sudan, and D. A. Wilson. Optimal Error Correction Against Computationally Bounded Noise. In *Theory of Cryptography, Second Theory of Cryptography Conference, Lecture Notes in Computer Science 3378*, pages 1–16. Springer-Verlag, February 2005.

[10] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology—Eurocrypt '99, Lecture Notes in Computer Science 1592*, pages 223–238. Springer-Verlag, 1999.

[11] L. Sassaman, B. Cohen, and N. Mathewson. The Pynchon Gate: A Secure Method of Pseudonymous Mail Retrieval. In *Workshop on Privacy in the Electronic Society (WPES 2005)*, pages 1–9, November 2005.

[12] L. Sassaman and B. Preneel. The Byzantine Postman Problem: A Trivial Attack against PIR-based Nym Servers. Technical Report ESAT-COSIC 2007-001, Katholieke Universiteit Leuven, February 2007.

[13] SciFace Software. Mathematics Mastered with MuPAD Pro. http://www.sciface.com/products/mupadpro.php.

[14] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, 1979.

[15] V. Shoup. NTL: A Library for doing Number Theory. http://shoup.net/ntl/.

[16] M. Tompa and H. Woll. How to Share a Secret with Cheaters. *J. Cryptology*, 1(2):133–138, 1988.

[17] E. Y. Yang, J. Xu, and K. H. Bennett. Private Information Retrieval in the Presence of Malicious Failures. In *Computer Software and Applications Conference 2002 (COMPSAC 2002)*, August 2002.

A Detailed proofs

Proof of Theorem 1. In step P2, the client defines r polynomials $f_1, \ldots, f_r$ of degree t. Define the s polynomials F_c to be $F_c = \sum_{1 \le j \le r} f_j W_{jc}$ for $1 \le c \le s$, where the W_{jc} are the words in the database as defined as in step S2. Note that these polynomials F_c are also of degree (at most) t, and also that $F_c(0) = \sum_{1 \le j \le r} f_j(0) W_{jc} = \sum_{1 \le j \le r} \delta_{j\beta} W_{jc} = W_{\beta c}$.

Suppose server i is one of the servers that responds. In step S1, it receives $[f_1(\alpha_i), \ldots, f_r(\alpha_i)]$ from the client. In step S3, it computes $R_{ic} = \sum_{1 \le j \le r} f_j(\alpha_i) W_{jc} = F_c(\alpha_i)$ for $1 \le c \le s$, and returns these values to the client in step S4.

The client initializes H to be $\{(G, \epsilon)\}$ in step C1, where G is the set of server numbers that responded. We claim that after x iterations of the loop at C4, H will be the set $\{(G, B_\beta^{(xw)})\}$, where $B_\beta^{(xw)}$ is a string consisting of the first xw bits of the database block B_β. We proceed by induction: we have already shown that this is the case for $x = 0$. Suppose it is true for $x = c - 1$ for some $1 \le c \le s$. Now consider iteration c. In step C5, the client calls $\textsc{EasyRecover}(\mathbb{S}, w, t, h, H, [R_{1c}, \ldots, R_{\ell c}], [\alpha_1, \ldots, \alpha_\ell])$ where $R_{jc} = F_c(\alpha_j)$ for each $j \in G$.

By the induction hypothesis, there is exactly one element of H, so the loop at E2 will execute only once. In step E4, the client will necessarily find $\phi = F_c$, since $\phi(\alpha_j) = R_{jc} = F_c(\alpha_j)$ for each $j \in G$, and so for each $j \in I \subseteq G$. Since these two polynomials ϕ and F_c of degree at most t agree on at least $t + 1$ points of $\mathbb{I}$, they must be equal. Therefore the W in step E5 will just be the w-bit representation of $F_c(0) = W_{\beta c}$, which is the c^{th} w-bit word of block β of the database.

Then in step E6, $\Gamma(\phi) \cap G$ will equal G, so H' will be set to $\{(G, \sigma || W)\}$. By the induction hypothesis, σ is the first $(c-1)w$ bits of B_β, so $\sigma || W$ is the first cw bits of B_β, and the proof of the claim is complete.

Therefore, after all $s = b/w$ iterations of the loop at C4, H will equal $\{(G, B_\beta)\}$, as required. □

Proof of Theorem 2. As above, we will prove that after x iterations of the loop at C4, H will be a set containing the element $(G_h^{(x)}, B_\beta^{(xw)})$ where $G_h^{(x)}$ is the set of server numbers that both replied at all, and also replied honestly in the first x words of its reply, and $B_\beta^{(xw)}$ is defined as before. For $x = 0$, this is trivially true. Suppose it is true for $x = c - 1$ for some $1 \le c \le s$. Now consider iteration c. We can assume the client calls $\textsc{HardRecover}$ in step C6, since as we noted earlier, $\textsc{EasyRecover}$ produces the same answer as $\textsc{HardRecover}$ when it produces an answer at all.

$\textsc{HardRecover}$ will produce a set of polynomials in step H2, one of which will necessarily be F_c (as defined above). This set may have polynomially (in k) many elements, but we will see in section 3.4 that the probability that this set contains elements other than the desired one can be made arbitrarily small, ensuring that the entire protocol runs in (probabilistic) polynomial time. When the loop at H3 encounters the element F_c, W_i will be set to $F_c(0) = W_{\beta c}$ (as defined above). By the induction hypothesis, step H5 will find the element $(G_h^{(c-1)}, B_\beta^{((c-1)w)}) \in H$. But also $\Gamma(F_c) \supseteq G_h$, so $\Gamma(F_c) \cap G_h^{(c-1)} \supseteq G_h$, so $|\Gamma(F_c) \cap G_h^{(c-1)}| \ge h$. Then step H5 will add $(\Gamma(F_c) \cap G_h^{(c-1)}, B_\beta^{((c-1)w)} || W_{\beta c}) = (G_h^{(c)}, B_\beta^{(cw)})$ to H', and the proof is complete. □

Proof of Fact 3. We start with the following Lemma:

Lemma. *There is a polynomial $\mathcal{P}_{\mathbb{S}}(k)$, depending only on whether $\mathbb{S}$ is a ring or a field, such that the size of the set of candidate polynomials $\{\phi_i\}$ output in step H2 is at most $\mathcal{P}_{\mathbb{S}}(k)$.*

Proof of Lemma. For fields $\mathbb{S}$, the algorithm of [7] works by constructing a bivariate polynomial $Q(x, y)$ over $\mathbb{S}$ with the property that for any (univariate) polynomial ϕ such that $|\Gamma(\phi)| \ge h$, it is the case that $(y - \phi(x))$ is a factor of Q. This polynomial Q has degree $\left\lfloor \frac{1}{t}\left(h - 1 + h\left\lfloor \frac{kt + \sqrt{(kt)^2 + 4(h^2 - kt)}}{2(h^2 - kt)} \right\rfloor\right)\right\rfloor$ in y, so the number of such factors is at most that value. (The denominator of this value is what produces the restriction that $h > \sqrt{kt}$.) With some simple algebra, it is easy to see that if $1 \le t < k$ and $\sqrt{kt} < h \le k$, that value is bounded by $2k^2$.

For rings $\mathbb{S} = \mathbb{Z}_{pq}$, we proceed modulo p and q separately, and combine the results using the Chinese Remainder Theorem, matching factors using $\Gamma(\phi)$. This can potentially increase the bound on the number of possible results to $(2k^2)^2$.

Note that neither of these bounds is tight. □

Each of the at most s calls to $\textsc{HardRecover}$ will produce a set of at most $\mathcal{P}_{\mathbb{S}}(k)$ polynomials in step H2. Each of these polynomials ϕ will have $|\Gamma(\phi)| \ge h$. But since h, the number of honest servers, is more than half the total number of servers that replied, at least some of the elements of $\Gamma(\phi)$ must be server numbers of honest servers. For each ϕ, there must be at least one such element, and for each ϕ other than the correct one, there must be at most t (otherwise, ϕ would agree with the correct polynomial in more than t places, and so it would indeed be the correct polynomial). The key observation, similar to that in [16], is that if the Byzantine servers cannot know the values of α_j for honest server numbers j, then they only have a small chance of producing incorrect polynomials that agree with the correct polynomial at one of the honest α_j.

Let Z be the set $\{\alpha_j : \text{server } j \text{ is not honest}\}$, and Y be the set $\{\alpha_j : \text{server } j \text{ is honest}\}$. We want to bound the probability that, of the $\leq s\mathcal{P}_{\mathbb{S}}(k)$ polynomials returned in step H2, and of the $\leq t$ points of $\mathbb{I}\backslash Z$ at which each of these polynomials agree with the correct polynomials, the resulting $\leq st\mathcal{P}_{\mathbb{S}}(k)$ points have non-trivial intersection with the set Y. But $|\mathbb{I}\backslash Z| = |\mathbb{I}| - (\ell - h)$, and $|Y| = h$, so that probability is at most $\frac{hst\mathcal{P}_{\mathbb{S}}(k)}{|\mathbb{I}|-\ell+h}$, as required. □

)07 IEEE Symposium on Security and Privacy(SP'07)
7695-2848-1/07 $20.00 © 2007 IEEE