Private Keyword-Based Push and Pull with Applications to Anonymous Communication

Extended Abstract

Lea Kissner[1], Alina Oprea[1], Michael K. Reiter[1,2], Dawn Song[1,2], and Ke Yang[1]

[1] Dept. of Computer Science, Carnegie Mellon University
{leak,alina,yangke}@cs.cmu.edu
[2] Dept, of Electrical and Computer Engineering, Carnegie Mellon University
{reiter,dawnsong}@cmu.edu

Abstract. We propose a new keyword-based Private Information Retrieval (PIR) model that allows private modification of the database from which information is requested. In our model, the database is distributed over n servers, any one of which can act as a transparent interface for clients. We present protocols that support operations for accessing data, focusing on privately appending labelled records to the database (push) and privately retrieving the next unseen record appended under a given label (pull). The communication complexity between the client and servers is independent of the number of records in the database (or more generally, the number of previous push and pull operations) and of the number of servers. Our scheme also supports access control oblivious to the database servers by implicitly including a public key in each push, so that only the party holding the private key can retrieve the record via pull. To our knowledge, this is the first system that achieves the following properties: private database modification, private retrieval of multiple records with the same keyword, and oblivious access control. We also provide a number of extensions to our protocols and, as a demonstrative application, an unlinkable anonymous communication service using them.

1 Introduction

Techniques by which a client can retrieve information from a database without exposing its query or the response to the database was initiated with the study of oblivious transfer [17]. In the past decade, this goal has been augmented with that of minimizing communication complexity between clients and servers, a problem labelled Private Information Retrieval (PIR) [8]. To date, PIR has received significant attention in the literature, but a number of practically important limitations remain: queries are limited to returning small items (typically single bits), data must be retrieved by address as opposed to by keyword search, and there is limited support for modifications to the database. Each of these limitations has received attention (e.g., [9,8,14,6]), but we are aware of no solution that fully addresses these simultaneously.

In this extended abstract we present novel protocols by which a client can privately access a distributed database. Our protocols address the above limitations while retaining privacy of queries (provided that at most a fixed threshold t of servers is compromised)

M. Jakobsson, M. Yung, J. Zhou (Eds.): ACNS 2004, LNCS 3089, pp. 16–30, 2004.

and while improving client-server communication efficiency over PIR solutions at the cost of server-server communication. Specifically, the operations we highlight here include:

- push In order to insert a new record into the database, the client performs a push operation that takes a label, the record data, and a public key as arguments.
- pull To retrieve a record, a client performs a pull operation with a label and a private key as arguments. The response to a pull indicates the number of records previously pushed with that label and a corresponding public key, and if any, returns the first such record that was not previously returned in a pull (or no record if they all were previously returned).

Intuitively, the pull operation functions as a type of "dequeue" operation or list iterator: each successive pull with the same label and private key will return a new record pushed with that label and corresponding public key, until these records are exhausted. We emphasize that the above operations are private, and thus we call this paradigm Private Push and Pull (P^3).

As an example application of these protocols, suppose we would like to construct a private bulletin board application. In this scenario, clients can deposit messages which are retrieved asynchronously by other clients. An important requirement is that the communication between senders and receivers remains hidden to the database servers, a property called *unlinkability*. Clients encrypt messages for privacy, and label them with a keyword, the mailbox address of the recipient. If multiple clients send messages to the same recipient, there exist multiple records in the database with the same keyword. We would like to provide the receiver with a mechansim to retrieve some or all the messages from his mailbox. Thus, the system should allow insertion and retrieval of multiple records with the same keyword. Another desirable property would be to provide *oblivious access control*, such that a receiver can retrieve from its mailbox only if he knows a certain private key. In addition, the database enforces the access control obliviously, i.e., the servers do not know the identity of the intended recipient. All these properties are achieved by our P^3 protocols and the construction of such a private bulletin board is an immediate application of these protocols.

Our protocols have additional properties. Labels in the database, arguments to push and pull requests, and responses to pull requests are computationally hidden from up to t maliciously corrupted servers and any number of corrupted clients. The communication complexity incurred by the client during a push or pull operation is independent of both the number of servers and the number of records in the database, and requires only a constant number of ciphertexts. While communication complexity between the servers is linearly dependent on both the number of servers and the number of records in the database, we believe that this tradeoff—i.e., minimizing client-server communication at the cost of server-server communication—is justified in scenarios involving bandwidth-limited or geographically distant clients.

Beyond our basic push and pull protocols, we will additionally provide a number of enhancements to our framework, such as: a peek protocol that, given a label and private key, privately retrieves the i-th record pushed with that label and corresponding public key; a modification to pull to permit the retrieval of arbitrary-length records; and the

ability to perform a pull based not only on identical label matching, but based on any predicate on labels (with additional cost in server-server communication complexity).

We define security of the P^3 protocols in the malicious and honest-but-curious adversary models. The definition of security that we employ is very similar to the definition of secure multi-party computation [11]. Proofs that P^3 satisfies the definition of security in the malicious adversary model will be given in the full version of the paper. We also propose a more efficient P^3 protocol that is secure in the honest-but-curious model. We thus achieve a tradeoff between the level of security guaranteed by our protocols and their computational complexity.

To summarize, the contributions of our paper are:

- The definition of a new keyword-based Private Information Retrieval model
  Our model extends previous work on PIR in several ways. Firstly, we enable private modification of the database, where the database servers do not learn the modified content. Secondly, we allow retrieval of a subset or all records matching a given keyword. And, finally, we provide *oblivious access control*, such that only the intended recipients can retrieve messages and the servers do not know the identity of message recipients.
- The construction of secure and efficient protocols in this model
  We design P^3 protocols, that achieve a constant communication complexity (in number of ciphertexts) between the clients and the servers and that are provably secure in the malicious adversary model.
- The design of an unlinkable [16] anonymous messaging service using the new proposed protocols
  The anonymous messaging service we design is analogous to a bulletin board, where clients deposit messages for other clients, to retrieve them at their convenience. The security properties of the P^3 protocols provide the system with unlinkability.

2 Related Work

As already mentioned, our P^3 primitive is related to other protocols for hiding what a client retrieves from a database. In this section we differentiate P^3 from these other protocols.

Private information retrieval (PIR) [9,8,3] enables a client holding an index $i, 1 \leq i \leq d$, to retrieve data item i from a d-item database without revealing i to the database. This can be trivially achieved by sending the entire database to the client, so PIR mandates sublinear (and ideally polylogarithmic) communication complexity as a function of d. Our approach relaxes this requirement for server-to-server communication (which is not typically employed in PIR solutions), and retains this requirement for communication with clients; our approach ensures client communication complexity that is *independent* of d. In addition, classic PIR does not address database changes and does not support labelled data on which clients can search.

Support for modifying the database was introduced in *private information storage* [14]. This supports both reads and writes, without revealing the address read or written. However, it requires the client to know the address it wants to read or write. P^3

eliminates the need for a client to know the address to read from, by allowing retrieval of data as selected by a predicate on labels. P^3 does not allow overwriting of values, but allows clients to retrieve all records matching a given query.

The problem of determining whether a keyword is present in a database without revealing the keyword (and again with communication sublinear in d) is addressed in [6]. The P^3 framework permits richer searches on keywords beyond identical matching—with commensurate additional expense in server complexity —though P^3 using identical keyword matching is a particularly efficient example. Another significant difference is that P^3 returns the data associated with the selected label, rather than merely testing for the existence of a label.

Also related to P^3 is work on *oblivious keyword search* [13], which enables a client to retrieve data for which the label identically matches a keyword. Like work on oblivious transfer that preceded it, this problem introduces the security requirement that the client learn nothing about the database other than the record retrieved. It also imposes weaker constraints on communication complexity. Specifically, communication complexity between a client and servers is permitted to be linear in d.

3 Preliminaries

A public-key cryptosystem is a triplet of probabilistic algorithms (G, E, D) running in expected polynomial time. $G(1^{\kappa}_{\mathcal{TE}})$ is a probabilistic algorithm that outputs a pair of keys (pk, sk), given as input a security parameter $\kappa_{\mathcal{TE}}$. Encryption, denoted as $E_{pk}(m)$, is a probabilistic algorithm that outputs a ciphertext c for a given plaintext m. The deterministic algorithm for decryption, denoted as $D_{sk}(c)$, outputs a decryption m of c. Correctness requires that for any message m, $D_{sk}(E_{pk}(m)) = m$.

The cryptosystems used in our protocols require some of the following properties:

- message indistinguishability under chosen plaintext attack (IND-CPA security) [12]: an adversary is given a public key pk, and chooses two messages m_0, m_1 from the plaintext space of the encryption scheme. These are given as input to a test oracle. The test oracle chooses $b \leftarrow_R \{0, 1\}$ and gives the adversary $E_{pk}(m_b)$. The adversary must not be able to guess b with probability more than negligibly different from $\frac{1}{2}$.
- (t, n) threshold decryption: a probabilistic polynomial-time (PPT) share-generation algorithm S, given pk, sk, t, n, outputs private shares $sk_1, \ldots, sk_n$ such that parties who possess at least $t + 1$ shares and a ciphertext c can interact to compute $D_{sk}(c)$. Specifically we require $(n - 1, n)$ threshold decryption, where the private shares are additive over the integers, such that $sk = \sum_{i=1}^{n} sk_i$.
- threshold IND-CPA security [10]: the definition for threshold IND-CPA security is the same as for normal IND-CPA security, with minor changes. Firstly, the adversary is allowed to choose up to t servers to corrupt, and observes all of their secret information, as well as controlling their behaviour. Secondly, the adversary has access to a partial decryption oracle, which takes a message m and outputs all n shares (constructed just as decryption proceeds) of the decryption of an encryption of m.
- partial homomorphism: there must be PPT algorithms $+_{pk}$, $-_{pk}$, $\cdot_{pk}$ for addition and subtraction of ciphertexts, and for the multiplication of a known constant by

a ciphertext such that for all a, b, in the plaintext domain of the encryption scheme, $c \in Z$, such that the result of the desired operation is also in the plaintext domain of the encryption scheme:

$$D_{sk}(E_{pk}(a) \ +_{pk} \ E_{pk}(b)) = a + b$$
$$D_{sk}(E_{pk}(a) \ -_{pk} \ E_{pk}(b)) = a - b$$
$$D_{sk}(c \ \cdot_{pk} \ E_{pk}(a)) = ca$$

- blinding: there must be a PPT algorithm Blind_{pk} which, given a ciphertext c which encrypts message m, produces an encryption of m, pulled from a distribution which is uniform over all possible encryptions of m.
- indistinguishability of ciphertexts under different keys (key privacy) [1]: the adversary is given two different public keys pk_0, pk_1 and it chooses a message from the plaintext range of the encryption scheme considered. Given an encryption of the message under one of the two keys, chosen at random, the adversary is not able to distinguish which key was used for encryption with probability non-negligibly higher than $\frac{1}{2}$.

3.1 Notation

- $a||b$ denotes the concatenation of a and b;
- $x \leftarrow D$ denotes that x is sampled from the distribution D;
- $\bar{x}$ denotes an encryption of x under an encryption scheme, that can be inferred from the context;
- $\mathcal{E} = (G, E, D)$, an IND-CPA secure, partially homomorphic encryption scheme, for which we can construct proofs of plaintext knowledge and blind ciphertexts. For the construction in Sec. 5, we also require the key privacy property. The security parameter for $\mathcal{E}$ is denoted as $\kappa_{\mathcal{E}}$.
- $\mathcal{TE} = (G^h, E^h, \mathsf{threshDecrypt})$, a threshold decryption scheme, which is threshold IND-CPA secure. $\mathsf{threshDecrypt}$ is a distributed algorithm, in which each party uses its share of the secret key to compute a share of the decryption. In addition, it should have the partial homomorphic property and we should be able to construct proofs of plaintext knowledge. The security parameter for $\mathcal{TE}$ is denoted as $\kappa_{\mathcal{TE}}$.
- $M_{pk}^{\mathcal{E}}$ denotes the plaintext space of the encryption scheme $\mathcal{E}$ for public key pk.
- $\Pi = \mathsf{zkp}[p]$ denotes the zero-knowledge proof of predicate p, $\Pi = \mathsf{zkpk}[p]$ denotes the zero-knowledge proof of knowledge of p

3.2 Paillier

The Paillier encryption scheme defined in [15] satisfies the first six defined properties. In the Paillier cryptosystem, the public key is an RSA-modulus N and a generator g that has an order a multiple of N in $\mathbb{Z}_{N^2}^*$. In order to encrypt a message $m \in \mathbb{Z}_N$, a random r is chosen in $\mathbb{Z}_N$, and the ciphertext is $c = g^m r^N \bmod N^2$. In this paper, we will consider the plaintext space for the public key (N, g) to be $M_{(N,g)} = (-\frac{N}{2}, \frac{N}{2})$ so that we can safely compute $-x$, given x in the plaintext space.

For the construction in Sec. 5, we need key privacy of the encryption scheme used. In order to achieve that, we slightly modify the Paillier scheme so that the ciphertext is $c + \mu N^2$, where μ is a random number less than a threshold $T = \frac{2^{4\kappa_{\mathcal{TE}}}}{N^2}$ ($\kappa_{\mathcal{TE}}$ is the security parameter).

The threshold Paillier scheme defined in [10] can be easily modified to use additive shares of the secret key over integers (as this implies shares over $N\lambda(N)$, and thus with the modification given above, satisfies the properties required for $\mathcal{TE}$.

The unmodified Pailler cryptosystem satisfies the requirements for $\mathcal{E}$. Zero-knowledge proofs of plaintext knowledge are given in [7].

3.3 System Model

We denote by n the number of servers, and t the maximum number that may be corrupted. Privacy of the protocols is preserved if $t < n$.

Assuming the servers may use a broadcast channel to communicate, every answer returned to a client will be correct if $t < n$ or all servers are honest-but-curious. This does not, however, guarantee that an answer will be given in response to every query. If every server may act arbitrarily maliciously (Byzantine failures), a broadcast channel may be simulated if $t < \frac{n}{3}$.

We do not address this issue in this paper, but liveness (answering every query) can be guaranteed with $t < \frac{n}{3}$ if every misbehaving server is identified and isolated, and the protocol is restarted without them. Note that this may take multiple restarts, as not every corrupted server must misbehave at the beginning.

In the malicious model, our protocols are simulatable [11], and thus the privacy of client queries, responses to those queries (including the presence or absence of information), and database records is preserved. In the honest-but-curious model, we may achieve this privacy property more efficiently. For lack of space, we defer the proofs to the full version of this paper.

The database supports two types of operations. In a push operation, a client provides a public key pk, a label ℓ, and data δ. In a pull operation, the client provides a secret key sk and a label x, and receives an integer and a data item in response. The integer should be equal to the number of previous push operations for which the label $\ell = x$ and for which the public key pk is the corresponding public key for sk. The returned data item should be that provided to the first such push operation that has not already been returned in a previous pull. If no such data item exists, then **none** is returned in its place.

4 The P^3 Protocol

We start the description of P^3 with the push protocol. Before going into the details of the pull protocol, we construct several building block protocols. We give several extensions to the basic protocols. We then analyze the communication complexity of the proposed protocols. At the end of the section, we suggest a more efficient implementation of our protocols in the honest-but-curious model.

In the protocols given in this paper, the selection predicate is equality of the given label x to the i^{th} record label ℓ_i, under a given secret key sk. This selection predicate is evaluated using the protocol testRecord. The P^3 system can be modified by replacing testRecord with a protocol that evaluates an arbitrary predicate, e.g., using [7].

4.1 Initial Service-Key Setup

During the initial setup of a P^3 system, the servers collectively generate a public/private key pair $(\mathsf{PK}, \mathsf{SK})$ for the threshold encryption scheme $\mathcal{TE}$, where PK is the public key, and the servers additively share the corresponding private key SK. We call the public/private key pair the system's *service key*. We require that $d < q, n \cdot d \cdot q^2 < 2^{\kappa_{\mathcal{T}\varepsilon}-1}$, $\frac{2^{2\kappa_\varepsilon+3}n}{n-t} < 2^{\kappa_{\mathcal{T}\varepsilon}-1}$, and $2^{\kappa_\varepsilon+1} + 3 \cdot 2^{2\kappa_\varepsilon+2} < 2^{\kappa_{\mathcal{T}\varepsilon}-1}$ so that the operations (presented next) over the message space $M_{pk}^{\mathcal{TE}}$ (which is an integer interval of length about $2^{\kappa_{\mathcal{T}\varepsilon}}$, centered around 0) will not "overflow". Here d denotes the number of records in the database, and q is a prime.

For notational clarity, the protocols are given under the assumption that the data sent to the server in a push operation can be represented as an element of $\mathbb{Z}_q$. This can be trivially extended to arbitrary length records (see 4.5).

4.2 The Private Push Protocol

When a client $\mathcal{C}$ wants to insert a new record in the distributed database, it first generates a public key/secret key pair (pk, sk) for the encryption scheme $\mathcal{TE}$ and then invokes a push operation $\mathsf{push}_{PK}(pk, \ell, \delta)$. Here PK is the service key, ℓ is the label and δ is the data to be inserted. The protocol is a very simple one and is given in Fig. 1. $\mathsf{H}(\cdot)$ is a cryptographically secure hash function, e.g., MD5.

Note that the data is sent directly to the server, and thus if privacy of the contents of the data is desired, the data should be encrypted beforehand.

$\mathsf{push}_{PK}(pk, \ell, \delta)$
Client $\mathcal{C}$ computes $y \leftarrow E_{pk}^h(\ell)$ and sends $\langle y, \mathsf{H}(\delta)||\delta\rangle$ together with a zero knowledge proof of knowledge $\Pi = \mathsf{zkpk}[l : l \in \mathbb{Z}_q, D_{sk}(y) = l]$.
This server adds the tuple $\langle y, \langle \mathsf{H}(\delta), \delta\rangle, E_{PK}^h(1)\rangle$ to the shared database.

Fig. 1. The push protocol

4.3 Building Block Protocols

The Decrypt Share Protocol. When the decryptShare protocol starts, one of the servers receives a ciphertext c encrypted using the public key pk of the threshold homomorphic encryption scheme $\mathcal{TE}$. It also receives an integer R representing a randomness range large enough to statistically hide the plaintext corresponding to c. We assume that the

servers additively share the secret key sk corresponding to pk, such that each server knows a share sk_i. After the protocol, the servers additively share the corresponding plaintext m. Each server will know a share m_i such that $\sum_{i=1}^{n} m_i = m$ and it will output a commitment of this share ($\bar{m}_i = E^h_{pk}(m_i)$). The protocol is given in Fig. 2 and is similar to the Additive Secret Sharing protocol in [7].

$\mathsf{decryptShare}_{sk_1,\ldots,sk_n}(c, R)$
We assume that an arbitrary server holds c — assume it is $\mathcal{S}_j$.

1. For $1 \leq i \leq n$, $\mathcal{S}_i$ chooses $a_i \leftarrow [0, \ldots, R]$, computes $c_i \leftarrow E^h_{pk}(a_i)$.
2. For $i = 1, \ldots, n$, $\mathcal{S}_i$ broadcasts c_i together with a zero knowledge proof of plaintext knowledge of c_i: $\Pi_i = \mathsf{zkpk}[a_i : a_i \in [0, \ldots, R], D_{sk}(c_i) = a_i]$.
3. All the servers check the zero knowledge proofs received from the other servers. If some proofs do not verify, then the servers that sent them are excluded from the protocol.
4. $\mathcal{S}_j$ computes $c' \leftarrow c +_{pk} c_1 +_{pk} c_2 +_{pk} \cdots +_{pk} c_n$.
5. All servers participate in $m' = \mathsf{threshDecrypt}_{sk_1,\ldots,sk_n}(c')$.
6. The additive share of m for $\mathcal{S}_j$ is $m_j = -a_j + m'$ and the commitment $\bar{m}_j$ can be computed as $\bar{m}_j = c' -_{pk} c_j$;
   The additive share of m for $\mathcal{S}_i$, $i \neq j$ is $m_i = -a_i$ and the commitment $\bar{m}_i$ can be computed as $\bar{m}_i = -_{pk} c_i$.

Fig. 2. The decryptShare protocol

The Multiplication Protocol. The mult protocol receives as input two encrypted values $\bar{x}$ and $\bar{y}$ under a public key pk of the threshold homomorphic encryption scheme $\mathcal{TE}$, and an integer R, used as a parameter to decryptShare. We assume that the servers additively share the secret key sk corresponding to pk, such that each server knows a share sk_i. The output of the protocol is a value $\bar{z}$ such that $D_{sk}(\bar{z}) = xy$. The protocol is given in Fig. 3 and is similar to the Mult protocol in [7].

$\mathsf{mult}_{pk}(\bar{x}, \bar{y}, R)$

1. All the servers participate in $\mathsf{decryptShare}_{sk_1,\ldots,sk_n}(\bar{y}, R)$, ending with additive shares of y: $y_1, \ldots, y_n$ and commitments of these shares $\bar{y}_1, \ldots, \bar{y}_n$.
2. For $1 \leq i \leq n$, $\mathcal{S}_i$ computes $\bar{t}_i = \bar{x} \cdot_{pk} y_i$ and broadcasts $\bar{t}_i$ together with a zero knowledge proof of knowledge $\Pi_i = \mathsf{zkpk}[y_i : D_{sk}(\bar{y}_i) = y_i, \bar{t}_i = \bar{x} \cdot_{pk} y_i]$.
3. All the servers check the zero knowledge proofs received from the other servers. If some proofs do not verify, then the servers that sent them are excluded from the protocol.
4. The output of the protocol is $\bar{z} = \bar{t}_1 +_{pk} \ldots +_{pk} \bar{t}_n$.

Fig. 3. The mult protocol

The Share Reduction Protocol. The shareModQ protocol receives as input a prime q, an encrypted value $\bar{x}$ under a public key pk of the threshold homomorphic encryption scheme $\mathcal{TE}$, and an integer R, used as a parameter to decryptShare. We assume that the servers additively share the secret key sk corresponding to pk, such that each server knows a share sk_i. The output of the protocol is $\bar{y}$ st $D_{sk}(\bar{y}) = D_{sk}(\bar{x})$, $\bar{y} = y_1 + \cdots + y_n, y_i \in \mathbb{Z}_q$. The protocol is given in Fig. 4.

shareModQ$_{pk}(\bar{x}, q, R)$

1. All the servers participate in decryptShare$_{sk_1,\ldots,sk_n}(\bar{x}, R)$, ending with additive shares of x: $x_1, \ldots, x_n$ and commitments of these shares $\bar{x}_1, \ldots, \bar{x}_n$.
2. For $1 \leq i \leq n$, $\mathcal{S}_i$ computes $y_i = x_i \bmod q$ and broadcasts $\bar{y}_i = E^h_{pk}(y_i)$ together with a zero knowledge proof of knowledge $\Pi_i = \mathsf{zkpk}[x_i, y_i : y_i \in \mathbb{Z}_q, D_{sk}(\bar{y}_i) = y_i, D_{sk}(\bar{x}_i) = x_i, y_i = x_i \bmod q]$.
3. All the servers check the zero knowledge proofs received from the other servers. If some proofs do not verify, then the servers that sent them are excluded from the protocol.
4. All the servers compute $\bar{y} = \bar{y}_1 \ +_{pk} \ \ldots \ +_{pk} \ \bar{y}_n$, which is the output of the protocol.

Fig. 4. The shareModQ protocol

The Modular Exponentiation Protocol. The expModQ protocol receives as input an encrypted value $\bar{x}$ under a public key pk of the threshold homomorphic encryption scheme $\mathcal{TE}$, an integer exponent k and a prime modulus q, and and an integer R, used as a parameter to decryptShare. The output of the protocol is $\bar{y}$ such that $D_{sk}(\bar{y}) = D_{sk}(\bar{x})^k$. In addition, the decryption of $\bar{y}$, y, can be written as $y = y_1 + \cdots + y_n$ with $y_i \in \mathbb{Z}_q$. We have thus the guarantee that $0 \leq y \leq (q-1)n$. The protocol is simply done by repeated squaring using the mult protocol. After each invocation of the mult protocol, a shareModQ protocol is executed.

4.4 The Private Pull Protocol

We have now all the necessary tools to proceed to the construction of the pull protocol. To retrieve the record associated with the label x encrypted under public key pk, the client $\mathcal{C}$ must know both x and the secret key sk corresponding pk. $\mathcal{C}$ encrypts both the label x and the secret key sk under the public service key PK and picks a public/secret key pair (pk', sk') for the encryption scheme $\mathcal{E}$. It then sends $\bar{x}$, $\bar{sk}$ and pk' to an arbitrary server.

Overview of the Pull Protocol. The servers will jointly compute a *template* $T = (T_1, \ldots, T_d)$, where d is the number of records in the database. The template is a series of indicators encrypted under pk', where T_i indicates whether x matches the label ℓ_i under sk (threshDecrypt$_{sk}(\ell_i) = x$) and whether i is the first record that matches ℓ_i not previously read. This determines whether it should be returned as a response to the

query ($D_{sk'}(T_i) = 1$) or not ($D_{sk'}(T_i) = 0 \bmod q$). The protocol returns to the client the template T and an encrypted counter, $\bar{m}$ that denotes the total number of records matching a given label.

The protocol starts in step 2 (Figure 5) with the servers getting additive shares of the secret key sk, sent encrypted by the client. In step 3, several flags are initialized, the meaning of which will be explained in Sec. 4.4. Then, in step 4, it performs an iteration on all the records in the database, calculating the template entry for each record. In steps 4(a)-4(e), for each record j in the database with the label encrypted under public key pk_j, a decryption under the supplied key sk and re-encryption of the label is calculated under the service public key PK. In order to construct the template, the additive homomorphic properties of the encryption scheme $\mathcal{TE}$ are used. For record j in the database, the servers jointly determine the correct template value (as explained above), using the building block testRecord.

The return result is constructed by first multiplying each entry in the template with the contents of the corresponding record, and then adding the resulting ciphertexts using the additive homomorphic operation $+_{pk'}$. At most one template value will hold an encryption of 1, so an encryption of the corresponding record will be returned. All other records will be multiplied by a multiple of q, and will thus be suppressed when the client performs $D_{sk'}(T) \bmod q$. The bounds on the size of the plaintext range ensure that the encrypted value does not leave the plaintext range.

An interesting observation is that our approach is very general and we could easily change the specification of the pull protocol, by just modifying the testRecord protocol. An example of this is given in Sec. 4.5, when we describe the peek protocol.

Flags for Repeated Keywords. In this section we address the situation in which multiple records are associated with the same keyword under a single key. The protocol employs a flag $\bar{f}$, which is set at the beginning of each pull invocation to an encryption of 1 under the public service key. $\bar{f}$ is obliviously set to an encryption of $0 \bmod q$ after processing the first record which both matches the label and has not been previously read. It will retain this value through the rest of the pull invocation. In addition, each record i in the database has an associated flag, $\bar{r}_i$. The decryption of $\bar{r}_i$ is 1 if record i has not yet been pulled and $0 \bmod q$ afterwards. Initially, during the push protocol, $\bar{r}_i$ is set to an encryption of 1.

The testRecord Protocol. The equality test protocol, testRecord, first computes $\bar{w}$ (steps 1-2), such that $\bar{1} \ -_{PK} \ \bar{w}$ is an encryption of 1 if $x = y \bmod q$ and an encryption of $0 \bmod q$ otherwise. In step 3, a flag $\bar{s}$ is computed as an encryption of 1 if the record matches the label, $f = 1$ (this is the first matching record), and $r = 1$ (this record has not been previously retrieved). We then convert $\bar{s}$ from an encryption under the service key PK to an encryption under the client's key pk of the same plaintext indicator ($0 \bmod q$ or 1). This is performed in steps 4-7 with result u. We then update the flags $\bar{f}$ and $\bar{r}$, as well as the counter $\bar{m}$. Both $\bar{r}$ and $\bar{f}$ are changed to encryptions of $0 \bmod q$ if the record will be returned in the pull protocol. The new value of $\bar{m}$ is obtained by homomorphically adding the match indicator $\bar{1} \ -_{PK} \ \bar{w}$ to the old value.

The detailed pull and testRecord protocols are given in Figs. 5 and 6.

$\mathsf{pull}(sk, x, pk', sk')$
The database, is a collection of d tuples $\{\mathcal{D}_j = \langle E^h_{pk_j}(\ell_j), e_j, \bar{r}_j = E^h_{PK}(r_j)\rangle\}_{j=1}^d$
Here $\ell_j \in Z_q$ and $e_j \in M_{pk'}$ can be parsed as $e_j = \mathsf{H}(\delta_j)||\delta_j$

1. $\mathcal{C}$ sends $(pk', \bar{x} = E^h_{\mathsf{PK}}(x), \bar{sk} = E^h_{\mathsf{PK}}(sk))$ to an arbitrary server $\mathcal{S}_j$, who broadcasts pk'.
2. All the servers participate in $\mathsf{decryptShare}_{SK_1,\ldots,SK_n}\left(\bar{sk}, \frac{2^{2\kappa\mathcal{E}+3}}{n-t}\right)$ and end with additive shares of sk: $sk_1, \ldots, sk_n$ and commitments $\bar{sk}_1, \ldots, \bar{sk}_n$.
3. An arbitrary sever computes $\bar{f} \leftarrow E^h_{\mathsf{PK}}(1), \bar{m} \leftarrow E_{pk'}(0)$ and broadcasts them to all the servers.
4. For $1 \le j \le d$, do:
   a) The server that holds $\mathcal{D}_j = \langle E^h_{pk_j}(\ell_j), e_j\rangle$ broadcasts it;
   b) All the servers participate in $\mathsf{decryptShare}_{sk_1,\ldots,sk_n}\left(E^h_{pk_j}(\ell_j), \frac{2q^2}{n-t}\right)$ and end with additive shares of ℓ'_j: $\ell'_{j1}, \ldots, \ell'_{jn}$ and commitments of these shares: $\bar{\ell}'_{j1}, \ldots, \bar{\ell}'_{jn}$ $(\ell'_j = \ell_j \Leftrightarrow sk = sk_j)$;
   c) Each server $\mathcal{S}_i$ broadcasts $\bar{y}_{ji} \leftarrow E^h_{PK}(\ell'_{ji})$, together with a zero knowledge proof of plaintext equality $\Pi_i = \mathsf{zkp}[y_{ji} : D_{SK}(\bar{y}_{ji}) = y_{ji}, D_{sk}(\bar{\ell}'_{ji}) = y_{ji}]$;
   d) All the servers check the zero knowledge proofs received from the other servers. If some proofs do not verify, then the servers that sent them are excluded from the protocol;
   e) All the servers compute $\bar{y}_j = \bar{y}_{j1} +_{PK} \ldots +_{PK} \bar{y}_{jn}$;
   f) All the servers participate in $\mathsf{testRecord}_{pk'}(\mathsf{PK}, \bar{x}, \bar{y}_j, \bar{f}, \bar{r}_j, \bar{m})$ to obtain $(T_j, \bar{f}, \bar{r}'_j, \bar{m})$.
   g) Set the database tuple $\mathcal{D}_j$ to be $\langle E^h_{pk_j}(\ell_j), e_j, \bar{r}'_j\rangle$.
   (the template is $(T_1, T_2, \ldots, T_d)$)
5. An arbitrary server computes $T = (T_1 \cdot_{pk'} e_1) +_{pk'} \cdots +_{pk'} (T_d \cdot_{pk'} e_d)$ and sends T and $\bar{m}$ to $\mathcal{C}$.
6. $\mathcal{C}$ computes $e \leftarrow D_{sk'}(T) \bmod q, m \leftarrow D_{sk'}(\bar{m}) \bmod q$ and parses e as $e = (r, \delta)$.
   - if $m = 0$, output none;
   - otherwise, check $r = \mathsf{H}(\delta)$ and if this holds, output data δ and m number of matches;
   - if consistency check does not hold, output error.

Fig. 5. The pull protocol

4.5 Extensions

Data of Arbitrary Length. The protocols given above can be extended to record data of arbitrary length as follows. First, the push operation can be naturally extended to include multiple data items, e.g., $\mathsf{push}(E_{pk}(\ell), \delta_1, ..., \delta_k)$. Next, step 4 in the pull protocol (Fig. 5) can be performed for each of the k data items, using the same template $(T_1, ..., T_d)$. Note that this does not increase the communication complexity among the servers. This is particularly efficient for large data records. For example, if the Paillier system is used, then the client/server communication complexity is asymptotically twice the actual data size transmitted.

$\mathsf{testRecord}_{pk}(\mathsf{PK}, \bar{x}, \bar{y}, \bar{f}, \bar{r}, \bar{m})$

1. All the servers participate in $\bar{z} \leftarrow \mathsf{shareModQ}_{PK}\left(\bar{x} \ -_{PK} \ \bar{y}, q, \frac{2q^2}{n-t}\right)$.
2. All the servers participate in $\bar{w} \leftarrow \mathsf{expModQ}_{PK}\left(\bar{z}, q-1, q, \frac{2q^2}{n-t}\right)$.
3. All the servers participate in $\bar{g} \leftarrow \mathsf{mult}_{PK}\left(\bar{1} \ -_{PK} \ \bar{w}, \bar{r}, \frac{2q^2}{n-t}\right)$ and $\bar{s} \leftarrow \mathsf{mult}_{PK}\left(\bar{g}, \bar{f}, \frac{2q^2}{n-t}\right)$.
4. All the servers participate in $\mathsf{decryptShare}_{SK_1,\ldots,SK_n}\left(\bar{s}, \frac{2(n-1)^2q^2}{n-t}\right)$ and end up with shares $s_1, \ldots, s_n$ and commitments $\bar{s}_1, \ldots, \bar{s}_n$.
5. $\mathcal{S}_i$ computes $u_i \leftarrow E_{pk}(s_i), i = 1, \ldots, n$. Then, $\mathcal{S}_i$ broadcasts u_i together with a zero knowledge proof $\Pi_i = \mathsf{zkp}[t_i : D_{sk}(u_i) = s_i, D_{SK}(\bar{s}_i) = s_i]$.
6. All the servers check the zero knowledge proofs received from the other servers. If some proofs do not verify, then the servers that sent them are excluded from the protocol.
7. All the servers compute $u = u_1 \ +_{pk} \ u_2 \ +_{pk} \ \cdots \ +_{pk} \ u_n$.
8. $\bar{r}' \leftarrow \mathsf{mult}_{PK}\left(\bar{r}, \bar{1} \ -_{PK} \ \bar{s}, \frac{2(n-1)^2q^2}{n-t}\right)$, $\bar{f}' \leftarrow \mathsf{mult}_{PK}\left(\bar{f}, \bar{1} \ -_{PK} \ \bar{g}, \frac{2(n-1)^2q^2}{n-t}\right)$.
9. The servers get a re-encryption of $1 - w$ under public key pk', analogously to steps 4-6 above. Denote the additive shares by $h_1, \ldots, h_n$ and the encryption of $1 - w$ under pk' by h. Then, the servers update $\bar{m}' \leftarrow \bar{m} \ +_{pk} \ h$.
10. The output of the protocol is the tuple $(u, \bar{f}', \bar{r}', \bar{m}')$.

Fig. 6. The testRecord protocol

The Peek Protocol. In order to retrieve a matching record by index, here we sketch a peek protocol, which can be easily derived from the pull protocol.

In addition to the parameters to the pull protocol, the peek protocol includes a flag $\bar{i}$, which is an encryption of the desired index i under the public service key. The database will return the i^{th} record matching label i or 0, if this does not exist, as well as the number of records matching the label. The flags $\bar{r}_j$ for each record and the flag $\bar{f}$ are not used in this version of the protocol. In step 4(f) the parameters passed to the testRecord protocol are $PK, \bar{x}, \bar{y}_j$, and $\bar{i}$. These are the only changes to the pull protocol.

The servers obliviously decrement $\bar{i}$ at each match found in the database, and return the record at which $\bar{i}$ becomes an encryption of 0. After steps 1-2 in testRecord, we test if $\bar{i}$ is an encryption of 0. We insert a step 2' after step 2, in which $\bar{e} \leftarrow \mathsf{expModQ}_{PK}\left(\bar{i}, q-1, q, \frac{2q^2}{n-t}\right)$ is computed. $\bar{1} \ -_{PK} \ \bar{e}$ is an encryption of 1 if $i = 0$. Step 3 changes to $\bar{t} \leftarrow \mathsf{mult}_{PK}\left(\bar{1} \ -_{PK} \ \bar{w}, \bar{1} \ -_{PK} \ \bar{e}, \frac{2(n-1)^2q^2}{n-t}\right)$. Steps 4-7 remain the same. In step 8, we update the value of the index to $\bar{i} \ -_{PK} \ (\bar{1} \ -_{PK} \ \bar{w})$.

Beyond Exact Label Matching. We have described our push and pull protocols in terms of exact label matching, though this can be generalized to support retrieval based on other predicates on labels. Specifically, given a common predicate Π, on a pull request with label x the servers could use secure multiparty computation (the techniques in [7]

are particularly suited in our setting) to compute the template $(T_1, \ldots, T_d)$ indicating the records for which the labels match x under predicate Π.

4.6 Efficiency

Our push, pull and peek protocols achieve a constant communication complexity in ciphertexts between the client and the servers. The communication among the servers in the pull protocol is proportional to the number of records in the distributed database and the number of servers.

We achieve a tradeoff between the level of security obtained by our protocols and their computational and communication complexity. If complexity is a concern, then more efficient protocols can be constructed by removing the zero-knowledge proofs and the value commitments generated in the protocols. Using standard techniques, we could show that the protocols constructed this way are secure in the honest-but-curious model. However, due to space limitations, we do not address this further in the paper.

5 Asynchronous Anonymous Communication

P^3 potentially has many uses in applications where privacy is important. As an example, in this section we outline the design of a simple anonymous message service using P^3 as a primitive. This message service enables a client to deposit a message for another client to retrieve at its convenience.

The messaging scheme is as follows:

- A sender uses the push protocol to add a label, encrypted under the receiver's public key, and a message to the database. In this context we call the label a *mailbox address*.
  - The message should be encrypted for privacy from the servers.
  - The mailbox address can either be a default address or one established by agreement between the sender and receiver. This agreement is necessary so that the receiver may retrieve the message.
- A receiver uses the pull or peek protocol to retrieve messages sent to a known mailbox address under his public key.

Because messages will accumulate at the servers, they may wish to determine some schedule on which to delete messages. Reasonable options include deleting all messages at set intervals, or deleting all messages of a certain set age.

Privacy. We achieve the content privacy and unlinkability anonymity properties as described in [9]. If the sender encrypts the message submitted to the servers, the servers cannot read the message, and thus achieves content privacy. Unlinkability concerns the ability for the servers to determine which pairs of users (if any) are communicating. As the P^3 servers can not determine the public key under which a label was encrypted, the label itself, or the text of the message, it has no advantage in determining the intended recipient of a message. Nor can they determine which message a client retrieved, if any, or even if a message has been retrieved by any client at any past time. Thus the servers

have no advantage in determining which client was the actual recipient of any given message.

As well as these properties, we achieve anonymity between senders and receivers. Any party may either retain this anonymity, or identify himself to other parties.

Senders are by default anonymous to receivers if they address their message to the default mailbox address. Note that the key with which they addressed their message is invisible to the recipient, and so a recipient cannot give a certain public key to a certain sender to abridge their anonymity. A sender may construct an anonymous return address, for use in addressing return messages, by encrypting an appropriate label under the sender's own public key. As we require key privacy of the cryptosystem used, the receiver cannot link the public key used to the identity of the sending party. A sender may sign their messages using a key to which they have attached an identity, if they do not wish to be anonymous.

Asynchronous Communication. Our system also benefits from the property of asynchrony, meaning that the senders and receivers do not have to be on-line simultaneously to communicate. The system is analogous to a bulletin board, where senders deposit messages and from which receivers retrieve them in a given interval of time. From this perspective, our system offers a different type of service than most prior approaches to anonymous communication (e.g., [4,16,5,19,18]) which anticipate the receiver being available when the sender sends. A notable exception is [9], which bears similarity to our approach. However, our use of P^3 permits better communication complexity between the clients and servers than does the use of PIR in [9].

6 Conclusion

We defined the Private Push and Pull (P^3) architecture. This allows clients to privately add (through the push protocol) and retrieve (through the pull or peek protocols) records in the database through transparent interaction with any of the distributed database servers. Under the protocols given, the servers identify which record is to be returned through keyword matching under a particular secret key. If at most t of n servers are actively corrupted, the keyword, key, and return result of a pull or peek protocol is computationally hidden from the servers, and any number of colluding clients.

Client communication in P^3 is independent of both the size of the database and the number of database servers, and requires only the number of ciphertexts corresponding to encryption of the data. Communication between the servers is linear in both the number of records in the database and the number of servers.

Using these protocols, we suggest an implementation of an anonymous messaging system. It achieves unlinkability, but both sender and receiver anonymity can be achieved through slight modifications.

References

1. M. Bellare, A. Boldyreva, A. Desai, D. Pointcheval. *Key-Privacy in Public Key Encryption*. In *Advances in Cryptology — Asiacrypt'01*, LNCS 2248.

2. M. Blum, A. De-Santis, S. Micali, G. Persiano. *Noninteractive Zero-Knowledge*. In *SIAM Journal on Computation*, vol. 20, pp. 1084-1118, 1991.
3. C. Cachin, S. Micali, M. Stadler. *Computational Private Information Retrieval with Polylogarithmic Communication*. In *Advances in Cryptology — Eurocrypt '97*, pp. 455-469, 1997.
4. D. Chaum. *Untraceable electronic mail, return addresses, and digital pseudonyms*. In *Communications of the ACM* 24(2):84–88, February 1981.
5. D. Chaum. *The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability*. In *Journal of Cryptology*, 1(1), pp 65-75, 1988.
6. B. Chor, N. Gilboa, M. Naor. *Private Information Retrieval by Keywords* Technical Report TR CS0917, Department of Computer Science, Technion, 1997
7. R. Cramer, I. Damgård, J. Buus Nielsen. *Multiparty Computation from Threshold Homomorphic Encryption*. In *Advances in Cryptology – Eurocrypt 2001*, pp. 280-299, 2001.
8. B. Chor, O. Goldreich, E. Kushilevitz, M. Sudan. *Private information retrieval*. In *Proc. 36th IEEE Symposium on Foundations of Computer Science*, 1995.
9. D. A. Cooper, K. P. Birman. *Preserving privacy in a network of mobile computers*. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, pages 26–38, May 1995.
10. P. Fouque, G. Poupard, J. Stern. *Sharing Decryption in the Context of Voting of Lotteries*. In *Financial Crypto 2000*, 2000.
11. O. Goldreich. *Secure Multi-Party Computation*. Working draft available at `http://theory.lcs.mit.edu/~oded/gmw.html`.
12. S. Goldwasser, S. Micali. *Probabilistic Encryption*. In *Journal of Computer and Systems Sciencee*, vol. 28, pp 270-299, 1984.
13. W. Ogata, K. Kurosawa. *Oblivious keyword search*. Available at `http://eprint.iacr.org/2002/182/`.
14. R. Ostrovsky, V. Shoup. *Private information storage*. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, 1997.
15. P. Paillier. *Public-key cryptosystems based on composite degree residue classes*. In *Advances in Cryptology – EUROCRYPT '99* (LNCS 1592), pp. 223–238, 1999.
16. A. Pfitzmann, M. Waidner. *Networks without user observability*. *Computers & Security* 2(6):158–166, 1987.
17. M. Rabin. *How to exchange secrets by oblivious transfer*. Technical Report, Tech. Memo. TR-81, Aiken Computation Laboratory, Harvard University, 1981.
18. M G. Reed, P. F. Syverson, D. M. Goldschlag. *Anonymous connections and onion routing*. *IEEE Journal on Selected Areas in Communication*, Special Issue on Copyright and Privacy Protection, 1998.
19. M. K. Reiter, A. D. Rubin. *Crowds: Anonymity for web transactions*. *ACM Transactions on Information and System Security* 1(1):66–92, November 1998.