Data & Knowledge Engineering 68 (2009) 1237–1252

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

User-private information retrieval based on a peer-to-peer community ☆

Josep Domingo-Ferrer *, Maria Bras-Amorós, Qianhong Wu, Jesús Manjón

Universitat Rovira i Virgili, UNESCO Chair in Data Privacy, Department of Computer Engineering and Mathematics, Av. Països Catalans 26, E-43007 Tarragona, Catalonia, Spain

ARTICLE INFO

Article history:
Available online 23 June 2009

Keywords:
Privacy in statistical databases
Private information retrieval
Combinatorial designs
Ring signatures

ABSTRACT

Private information retrieval (PIR) is normally modeled as a game between two players: a user and a database. The user wants to retrieve some item from the database without the latter learning which item is retrieved. Most current PIR protocols are ill-suited to provide PIR from a search engine or large database: (i) their computational complexity is linear in the size of the database; (ii) they assume active cooperation by the database server in the PIR protocol. If the database cannot be assumed to cooperate, a peer-to-peer (P2P) user community is a natural alternative to achieve some query anonymity: a user gets her queries submitted on her behalf by other users in the P2P community. In this way, the database still learns which item is being retrieved, but it cannot obtain the real query histories of users, which become diffused among the peer users. We name this relaxation of PIR user-private information retrieval (UPIR). A peer-to-peer UPIR system is described in this paper which relies on an underlying combinatorial structure to reduce the required key material and increase availability. Extensive simulation results are reported and a distributed key management version of the system is described.

1. Introduction

In private information retrieval (PIR), a user wants to retrieve an item from a database or search engine without the latter learning which item the user is retrieving. This is often a neglected aspect when designing data access frameworks [17]. PIR was invented in 1995 by Chor et al. [5,6] with the assumption that there are at least two copies of the same database, which do not communicate with each other. In the same paper, Chor et al. showed that single-database PIR (that is, with a single copy) is infeasible in the information-theoretic sense. However, two years later, Kushilevitz and Ostrovsky [16] presented a method for constructing single-database PIR based on the algebraic properties of the Goldwasser–Micali public-key encryption scheme [9]. Subsequent developments in PIR are surveyed in [22].

In the PIR literature the database is usually modeled as a vector. The user wishes to retrieve the value of the *i*th component of the vector while keeping the index *i* hidden from the database. Thus, it is assumed that the user knows the physical address of the sought item, which might be too strong an assumption in many practical situations. Keyword PIR [4] is a more

☆ The authors are with the UNESCO Chair in Data Privacy, but they are solely responsible for the views expressed in this paper, which do not necessarily reflect the position of UNESCO nor commit that organization. We are indebted to three anonymous reviewers, whose comments helped us to improve this paper. This work was partly supported by the Spanish Government through projects TSI2007-65406-C03-01 "E-AEGIS" and CONSOLIDER INGENIO 2010 CSD2007-00004 "ARES", and by the Government of Catalonia under grant 2009 SGR 1135. The first author is partly supported as an ICREA-Acadèmia researcher by the Government of Catalonia.

* Corresponding author.
E-mail addresses: josep.domingo@urv.cat (J. Domingo-Ferrer), maria.bras@urv.cat (M. Bras-Amorós), qianhong.wu@urv.cat (Q. Wu), jesus.manjon@urv.cat (J. Manjón).

0169-023X/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2009.06.004

flexible form of PIR: the user can submit a query consisting of a keyword and no modification in the structure of the database is needed.

We claim that PIR protocols proposed so far have two fundamental shortcomings which hinder their practical deployment:

(1) The database is assumed to contain n items and PIR protocols attempt to guarantee maximum privacy, that is, maximum server uncertainty on the index i of the record retrieved by the user. Thus, the computational complexity of such PIR protocols is $O(n)$, as proven in [5,6]. Intuitively, all records in the database must be "touched"; otherwise, the server could rule out some of the records when trying to discover i. For large databases, an $O(n)$ computational cost is unaffordable [2].
(2) It is assumed that the database server cooperates in the PIR protocol. However, it is the user who is interested in her own privacy, whereas the motivation for the database server is dubious. Actually, PIR is likely to be unattractive to most companies running queryable databases, as it limits their profiling ability. This probably explains why no real instances of PIR-enabled databases can be mentioned.

If one wishes to run PIR against a search engine, there is another shortcoming beyond the lack of server cooperation: the database cannot be modeled as a vector in which the user can be assumed to know the physical location of the keyword sought. Even keyword PIR does not really fit, as it still assumes a mapping between individual keywords and physical addresses (in fact, each keyword is used as an alias of a physical address). A search engine allowing only searches of individual keywords stored in this way would be much more limited than real engines like Google or Yahoo.

In view of the above, relaxations of PIR seem necessary in order to attain practical systems offering some privacy in information retrieval. In [8], a system named Goopir is proposed in which a user masks her target query by ORing them with $k-1$ fake queries and then submits the resulting masked query to a search engine or large database which does not need to cooperate (in fact, it does not even need to know that the user is trying to protect her privacy). Strictly speaking, Goopir does not achieve PIR as defined above; rather, it provides $h(k)$-private information retrieval, in that it cloaks the target query within a set of k queries of entropy at least $h(k)$. This system works fine but it assumes that the frequencies of keywords and phrases that can appear in a query are known and available: for maximum privacy, the frequencies of the target and the fake queries should be similar, so that the uncertainty $h(k)$ of the search engine about the real target query is maximum. TrackMeNot [14] is another practical system based on a different principle: rather than submitting a single masked query for each actual query like Goopir, a browser extension installed in the user's computer hides the user's actual queries in a cloud of automatic "ghost" queries submitted to popular search engines at different time intervals. While practical at a small scale, if the use of TrackMeNot became generalized, the overhead introduced by ghost queries would significantly degrade the performance of search engines and communications networks. Also, the submission timing of automatic ghost queries may be distinguishable from the submission timing of actual queries, which could provide an intruder with clues to identify the latter type of queries.

1.1. Contribution and plan of this paper

Like [8,14], we propose to relax strict PIR in order to obtain a practical system. However, rather than cloaking a query in a set of queries in a standalone fashion, we propose here to cloak the user's query history in a peer-to-peer user community: a user gets her queries submitted on her behalf by other users in the P2P community. In this way, the database still learns which item is being retrieved (which deviates from strict PIR), but it cannot obtain the real query histories of users, which become diffused among the peer users. We name the resulting PIR relaxation user-private information retrieval (UPIR). This approach certainly requires the availability of peers, not needed in the standalone systems [8,14], but it has some advantages: unlike [8], it does not require knowledge of the frequencies of all possible keywords and phrases that can be queried; unlike [14], it avoids the overhead of ghost query submission.

Note that what we offer is different from what can be achieved using anonymization systems based on onion routing, like Tor [27]. In an onion routing system, the transport of data is protected by bouncing the communication between a user and a server around a distributed network of volunteer relays, with a view to protecting against traffic analysis. However, such systems give no end-to-end protection (at the application level). Specifically, as long as a search engine (or a database server) can link the successive queries submitted by the same user (*e.g.* by using cookies or some other mechanism), the profiling and the re-identification capabilities of the search engine are unaffected even if the user is submitting her queries through Tor: the user still submits all of her queries herself (the relays merely relay them), so her query history is unaltered and a query history may suffice for re-identification, as illustrated by the AOL query disclosure scandal in August 2006 [1]. What we propose is to diffuse a user's query profile among the peers in a peer-to-peer community. However, onion routing systems can indeed complement our solution and be used for peers to communicate among themselves and hide their identity from each other at the transport level.

The new scheme uses a type of combinatorial design called configuration to increase service availability and reduce the number of required keys (see [24,18] for background on designs and configurations). The use of configurations in cryptographic key management is not new (*e.g.* see [18]), but their use in private information retrieval is.

Section 2 presents two simple peer-to-peer UPIR protocols and uses their shortcomings to motivate the use of configurations. Section 3 gives background on configurations, and then contributes an algorithm to search them and two constructions of larger configurations from smaller ones. Section 4 describes the proposed peer-to-peer UPIR protocol. Section 5 assesses the performance and the privacy offered by the protocol. Section 6 reports simulation results for the protocol. Section 7 shows how the protocol can be modified so that no trusted dealer is needed to run the configuration-based key management. Finally, Section 8 sketches conclusions and future work.

A preliminary and partial version of this work was presented in the conference paper [7]. Beyond extending this introduction and rewriting Section 3.1, the following new work has been added specifically for this journal paper: Section 2 (motivation of the use of configurations for P2P UPIR), Section 3.2 (constructions of larger configurations from smaller ones), Section 6 (simulation results) and Section 7 (dealer-free extension).

2. Peer-to-peer UPIR and configurations

Consider a peer-to-peer (P2P) community consisting of b users, in which the users submit queries on behalf of other users. The primary goal is to prevent the database or search engine from obtaining the query profile of a specific user. The secondary goal is for each user to preserve as much as possible the privacy of her query profile in front of the rest of users.

In the above setting, users do not need to know each other's identity. When deciding whether identities are to remain pseudonymous or not, one should carefully ponder whether the increased mutual trust derived from mutual knowledge compensates the loss of privacy of users in front of the rest of users in the P2P community. We will henceforth assume user pseudonymity.

We will next present two basic P2P protocols for UPIR. Their shortcomings will be analyzed, which will motivate the need for a more sophisticated protocol. The following assumptions are made for both protocols and for those in the rest of this article:

- Memory sectors shared by a group of users are used where the latter can record their queries, read other user's queries, record the database answers to queries submitted on behalf of other users and read the database answers to each user's own queries[1];
- Information is stored by users of a shared memory sector encrypted under an appropriate key of a symmetric cipher (*e.g.* see [20]); encryption protects the confidentiality of queries and answers in front of third parties not sharing the symmetric key (*e.g.* the database server or any user different from the one originating the query or from those who are authorized to submit it on behalf of the former);
- When a user decrypts a shared memory sector, she can distinguish the decrypted queries and answers to queries from garbage; some kind of redundancy (*e.g* a cyclic redundancy check) can be appended to the query or the query answer to facilitate this distinction.
- Our adversary model considers three types of adversaries to the query privacy of a specific user u_i:
  - The database, who receives in cleartext the queries of all users, including those of u_i;
  - The rest of users in the P2P community, who share one or more symmetric keys with u_i and who can read the queries and answers in the corresponding shared memory sectors; we assume that peers sharing keys with u_i correctly follow the protocols but can be curious, that is, we assume they are semi-honest;
  - External intruders, who do not fall into the above categories but want to compromise the privacy of users.

2.1. All-to-all protocol

Protocol 1 below uses a single memory sector m shared by all b users in the community, who also share a common encryption key x.

Protocol 1 (All-to-all P2P UPIR(q_i))

(1) In order to submit a query q_i to a database or search engine, user u_i first reads the shared memory sector m and decrypts it under x. Five cases can arise depending on the outcome of decryption:
  (a) The outcome is garbage, which means that memory sector m is free. In this case, u_i encrypts q_i under x and records the encrypted query in sector m.
  (b) The outcome is a query q_j issued by some other user in the community. In this case, u_i submits q_j to the database/search engine and records in sector m the answer obtained after encrypting it under key x. Thereafter, u_i waits a random (short) time and then goes back to Step 1 to obtain assistance in the submission of her own query q_i.
  (c) The outcome is the answer to a previous query q'_j issued by some other user u_j and previously submitted by a user $u_{j'}$ to the database/search engine on behalf of u_j. Since this answer has not yet been read by u_j (a user is assumed to erase the query answer when she reads it), u_i waits a random (short) time and then goes back to Step 1 to obtain assistance in the submission of her own query q_i.

[1] A simple wiki-like collaborative environment can be used to implement a shared memory sector. One may further assume that users access the wiki using some kind of onion routing protocol, in order to guard against traffic analysis by other peers or external intruders.

(d) The outcome is a query q_i' previously issued by u_i, who expects some other user to submit it on u_i's behalf. Since there is a previous query pending to be serviced by some other user, u_i waits a random (short) time and goes back to Step 1 to obtain assistance with the submission of her new query q_i.
(e) The outcome is the answer to a previous query q_i' issued by u_i and previously submitted by some other user to the database/search engine on behalf of u_i. In this case, u_i reads the answer, then encrypts her new query under key x and finally records the encrypted query in sector m.

Protocol 1 iterates until u_i manages to submit her query q_i. The random delay waited by u_i before retrying access to the shared memory sector in Steps b, c and d above increases with the number of failed retries, and it is inspired by the random backoff period used to handle access collisions in Ethernet-like local area networks (*e.g.*, see [26]); therefore, this mechanism gives reasonable assurance that u_i will eventually be able to submit her query. Also, in order to prevent the shared memory sector from being indefinitely jammed at Step 1c by a specific user u_j not collecting the answer to a previously submitted query, a timeout can be imposed on how long the shared memory sector must hold a certain query answer before the latter can be overwritten (by u_i or another user). Still, jamming might happen at Step 1d if no other user submits q_i', but this would imply that u_i is the only active user in the system; a countermeasure against such a possibility is to rely on a large community, that is, to take a large b.

Once u_i has used Protocol 1 to submit her query q_i, u_i must keep frequently calling the protocol with a garbage query as argument until she can collect the answer to q_i. Note that calling Protocol 1 with a garbage query reduces the protocol to Steps d and e, and causes Step e to free the shared memory sector m (u_i records the garbage query in m).

Let us now examine the privacy properties of Protocol 1:

- *Privacy in front of the database*. The protocol performs well, because the query profile of a user u_i is diffused among all remaining $b-1$ users.
- *Privacy in front of the remaining users*. All queries being submitted by a user can be read by all users in the P2P community, although in principle the latter do not know the identity of the former user. However, even if users are pseudonymous, the availability of side information (like the IP addresses of users accessing the shared memory sector or the very particular interests of a certain user) could allow linking all queries by the same u_i, with the subsequent profiling and re-identification risk for u_i. *E.g.* a way to link u_i's queries is through her IP address when u_i writes her queries or reads her query answers (an onion routing system like Tor could help to mitigate this risk, though).
- *Privacy in front of external intruders*. In principle, external intruders do not know the encryption key x, so they cannot see the queries submitted by u_i. However, since x is shared by all users in the P2P community, leakage of x to external intruders is more likely than if x was only shared by two users.

2.2. One-to-one protocol

Protocol 2 below uses a different shared memory sector m_{ij} and a different shared key x_{ij} for each pair of users (u_i, u_j).

Protocol 2 (One-to-one P2P UPIR)(q_i)

(1) In order to submit a query q_i to a database or search engine, user u_i randomly selects one of the $b-1$ remaining users. Let j be the selected user.
(2) u_i reads the memory sector m_{ij} corresponding to key x_{ij} and decrypts it under x_{ij}. Five cases can arise depending on the outcome of decryption:
  (a) The outcome is garbage. In this case, u_i encrypts q_i using a symmetric cipher keyed by x_{ij} and records the encrypted query in sector m_{ij}.
  (b) The outcome is a query q_j issued by user u_j, who expects u_i to submit it on her behalf. In this case, u_i submits q_j to the database/search engine and records in sector m_{ij} the answer obtained after encrypting it under key x_{ij}. Thereafter, u_i goes back to Step 1 to select a new key and obtain assistance in submitting q_i to the database/search engine from some other user.
  (c) The outcome is the answer to a previous query q_j' issued by u_j and previously submitted by u_i to the database/search engine on behalf of u_j. Since this answer has not yet been read by u_j (a user is assumed to erase the query answer when she reads it), u_i goes back to Step 1 to select a new key and obtain assistance in submitting q_i to the database/search engine from some other user.
  (d) The outcome is a query q_i' previously issued by u_i, who expects u_j to submit it on u_i's behalf. Since there is a previous query pending to be serviced by u_j, u_i goes back to Step 1 to select a new key and obtain assistance in submitting q_i to the database/search engine from some other user.
  (e) The outcome is the answer to a previous query q_i' issued by u_i and previously submitted by u_j to the database/search engine on behalf of u_i. In this case, u_i reads the answer, then encrypts her new query under key x_{ij} and finally records the encrypted query in sector m_{ij}.

Protocol 2 iterates until u_i manages to submit her query. For u_i to fail in submitting her query, it should happen that either she is the only active user (failure at Step 1d above) or that the other active users do not read the answers to the queries submitted by u_i on their behalf (failure at Step 1c above). The likelihood of failure at Step 1d is minimized by choosing a large P2P community, that is, a large b; failure at Step 1c can be countered by imposing a timeout after which an unread answer can be overwritten by u_i with her own query.

Like in Protocol 1 above, u_i must keep frequently calling Protocol 2 with a garbage query and fixed selected user u_j until she can collect the answer to q_i.

If we look at the privacy properties of Protocol 2:

- *Privacy in front of the database*. The protocol is good by this criterion, because the query profile of a user u_i is diffused among all remaining $b-1$ users.
- *Privacy in front of the remaining users*. The protocol has the advantage over Protocol 1 that each remaining user u_j other than u_i only sees a fraction $1/(b-1)$ of the queries issued by u_i. However, there is a disadvantage too, because u_j knows with certainty that those queries were issued by u_i. Even if u_i is pseudonymous, after u_j can link a number of queries issued by u_i, the latter's identity is likely to be disclosed. This did not happen with Protocol 1, where side information was needed in order to link the successive queries issued by the same user.
- *Privacy in front of external intruders*. External intruders do not know the encryption keys x_{ij}, so they cannot see the queries submitted by u_i. On the other hand, since each key x_{ij} is shared only by u_i and *one* user u_j in the P2P community, the probability of key leakage to external intruders is less than in Protocol 1.

As to the amount of secret key material required, Protocol 2 certainly leaves room for improvement. Indeed, the protocol requires a high number of shared secret keys, one for each pair of users, that is $b(b-1)/2$ in all.

A last major drawback of Protocol 2 is its slow performance: after u_i records her query q_i in the sector m_{ij} shared with u_j, u_i must wait for u_j to read that sector and submit q_i. Note that only u_j can do that, so it may take quite a long time before u_j happens to read m_{ij}.

After having discussed the strengths and weaknesses of Protocols 1 and 2, our goal will be to find a solution minimizing their weaknesses while retaining as much as possible their attractive features, namely: (i) fast performance (Protocol 1); (ii) limited visibility to other users of queries issued by u_i (Protocol 2); (iii) reduced amount of key material (we will try to use substantially less than the $b(b-1)/2$ keys of Protocol 2, although probably more than the single key of Protocol 1); (iv) good protection in front of external intruders (Protocol 2).

We take combinatorial configurations as a building block for our solution. As explained in Section 3, configurations allow distributing a number v of keys among b users, with $1 \leqslant v \leqslant b(b-1)/2$, in such a way that each user gets the same number of keys and each key is shared by the same number of users. Clearly, the closer v to 1, the closer we are to the situation of Protocol 1, with a single key for all users; on the other hand, the closer v to $b(b-1)/2$, the closer we are to the situation of Protocol 2, with a different key for each user pair. We will explore the intermediate situations and their trade-offs in the rest of this paper.

3. (v, b, r, k)-Configurations: background and construction

We first define a combinatorial design and then a configuration as a special type of design.

Definition 1 (*Design*). A *design* is a pair $(X, \mathscr{A})$, where X is a set of points and $\mathscr{A}$ is a finite set of subsets of X, called blocks. The degree of a point $x \in X$ is the number of blocks containing x. The rank of $(X, \mathscr{A})$ is the size of the largest block.

A design is said to be *regular* if all points have the same degree, say r. A design is said to be *uniform* if all blocks have the same size, say k (in which case the design is uniform of rank k). In the next definition we used the notations in [24,18].

Definition 2 (*(v,b,r,k)-1-design*). A (v,b,r,k)-1-design is a regular and uniform design with $|X| = v, |\mathscr{A}| = b$, degree r and rank k.

A (v,b,r,k)-1-design corresponds to a bipartite semiregular graph with $v+b$ vertices and degrees r and k. A necessary and sufficient condition for the existence of a (v,b,r,k)-1-design is that

$$bk = vr. \tag{1}$$

Definition 3 (*(v,b,r,k)-configuration*). A (v,b,r,k)-configuration is a (v,b,r,k)-1-design where any two distinct blocks intersect in zero or one point.

A (v,b,r,k)-configuration corresponds to a bipartite semiregular graph with $v+b$ vertices, degrees r and k, and girth strictly larger than 4. Configurations and their history have largely been studied by Gropp in [10–13].

The following lemma (an adaptation to configurations of a more general result in [18] on (v,b,r,k)-1-designs) quantifies the "connectivity" between blocks in a configuration.

Lemma 1. *In a (v,b,r,k)-configuration the number of blocks intersecting any specific block is $k(r-1)$.*

Proof. Consider a (v, b, r, k)-configuration $(X, \mathscr{A})$ and fix a block $A_i \in \mathscr{A}$. For any $x \in A_i$ define

$$\mathscr{B}_x = \{A_j \in \mathscr{A} : x \in A_j\} \setminus \{A_i\}.$$

Clearly, $|\mathscr{B}_x| = r - 1$ for all $x \in A_i$. On the other hand, the sets $\mathscr{B}_x (x \in A_i)$ are disjoint. Thus, the number of blocks intersecting A_i can be computed as

$$\left|\bigcup_{x \in A_i} \mathscr{B}_x\right| = \sum_{x \in A_i} |\mathscr{B}_x| = k(r-1). \qquad \square$$

A necessary condition for the existence of a (v, b, r, k)-configuration is $v \geqslant r(k-1) + 1$ [13]. Yet, this condition may not be sufficient.

The following is an example of a configuration:

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

3.1. A greedy algorithm to find configurations

Finding configurations and even determining whether a configuration with a given set of parameters exists is not trivial. We propose the next greedy algorithm to find a (v, b, r, k)-configuration if one exists. In what follows we label the points in X as integers from 1 to v.

We think of $\mathscr{A}$ as a list of b blocks, where a block is a list of k positions to which points in X must be assigned. Initially, all positions in all blocks of $\mathscr{A}$ are empty (*NULL*). For each block, the points in X are tried in sequence (from 1 to v) to fill the k block positions successively, with the aim of finding an assignment of points to positions which does not violate the configuration structure (see below for a description of what a violation is). We use a pair of indices (i, j), where i and j, respectively, to indicate which block in $\mathscr{A}$ and what position in the block we are attempting to fill:

- We start with $(i, j) = (1, 1)$, that is, we start by filling the first block at its first position.
- We then proceed by filling the ith block in $\mathscr{A}$ at its $(j+1)$th position while $j + 1 \leqslant k$, or by filling the first position of the $(i+1)$th block otherwise.

It can happen that no point can be found to fill the jth position of the current (ith) block which does not violate the requirements of a configuration, namely:

Correctness: No pair of points in X should be present in more than one block in $\mathscr{A}$.

Point availability: The candidate point to fill position j in the ith block should not have a label greater than $v - k + j$. Since points are tried in sequence, a candidate with label greater than $v - k + j$ for position j would imply that at most $k - j$ points with labels in $\{v - k + j + 1, \ldots, v\}$ would be left to fill the $k - j + 1$ remaining positions $\{j, \ldots, k\}$ in the block; hence, there would not be enough points left to fill those positions.

As long as the candidate point for the current assignment only violates the correctness requirement, the next point can be tried. If the point availability requirement is violated as well, then we must backtrack, *i.e.* reconsider the previous assignment. If the failed current assignment was the first one (that is, $j = 1$) of the ith block, backtracking means recomputing the kth assignment of the $i - 1$th block; if the failed assignment was not the first one (that is, $j > 1$), backtracking means recomputing the $j - 1$th assignment of the ith block.

If no configuration with the given parameters v, b, k parameters exists, the algorithm backtracks until $i = 0$ (failure). If a configuration exists, the algorithm will end up filling all b blocks appropriately.

As shown by the above description, the algorithm is correct, because it does nothing else than looking for an assignment of points to blocks which satisfies the definition of configuration. However, its computational complexity is exponential,

since backtracking is used to conduct an exhaustive search for a configuration with the required parameters. Thus, the algorithm can only be used to find small configurations. Its pseudocode is given in Algorithm 1.

Algorithm 1. Greedy configuration(v,b,k)

Require: Points to be assigned are labeled from 1 to v; b blocks with k positions each must be filled; $A_{i,j}$ is the point assigned to the jth position of the ith block; *cand* denotes the candidate point to fill a position.
Ensure: A (v,b,r,k)-configuration if one exists.
1: $(i,j) := (1,1)$ {We start with $A_{1,1}$}
2: **while** $0 < i \leqslant b$ **do**
3: $cand := \begin{cases} 1 & \textbf{if } A_{i,j} = NULL \text{ and } j = 1 \ \{A_{i,1} \leftarrow \text{point1}\} \\ A_{i,j-1}+1 & \textbf{if } A_{i,j} = NULL \text{ and } j > 1 \ \{\text{Try } A_{i,j} \leftarrow A_{i,j-1}+1\} \\ A_{i,j}+1 & \textbf{if } A_{i,j} \neq NULL \ \{\text{We come from a backtrack, so try next point}\} \end{cases}$
4: **while** $cand \leqslant v-k+j$ and assigning point *cand* to position $A_{i,j}$ violates the configuration correctness **do**
5: $cand := cand + 1$ {Keep seeking a right *cand*}
6: **end while**
7: **if** $cand = v-k+j+1$ **then**
8: $A_{i,j} := NULL$ {Not enough points left, so undo and backtrack}
9: $(i,j) := \begin{cases} (i-1,k) & \textbf{if } j = 1 \\ (i,j-1) & \textbf{if } j > 1 \end{cases}$
10: **else**
11: $A_{i,j} := cand$ {Assign *cand* and proceed}
12: $(i,j) := \begin{cases} (i,j+1) & \textbf{if } j < k \\ (i+1,1) & \textbf{if } j = k \end{cases}$
13: **end if**
14: **end while**
15: **if** $i = 0$ **then**
16: output $\emptyset$ {No configuration found}
17: **end if**
18: **if** $i = b+1$ **then**
19: output $\mathscr{A}$ {Configuration found}
20: **end if**

3.2. Building larger configurations from smaller ones

One problem when using configurations is the limited number of known configurations. We refer the reader to [13] for tables of parameters for which it is known that configurations exist and for which it is known that they do not exist. A consequence is that Algorithm 1 may fail to find a configuration with the required parameters. Worse yet, due to its greedy nature, the algorithm may take a very long time before it can decide whether the configuration exists or not.

On the good side, the projective planes over finite fields give us examples of (d^2-d+1, d^2-d+1, d, d)-configurations for any integer d such that $d-1$ is a power of a prime. See [25] for a very simple construction to actually find those configurations when $d-1$ is a prime. Additionally, given configurations (taken from the literature, obtained with the construction in [25], or found with Algorithm 1 if they are small), we show below how to easily construct larger configurations. Those new constructions are helpful to reduce the need for Algorithm 1.

3.2.1. Combining two not necessarily equal configurations

Suppose we have a (v,b,r,k)-configuration $(X = \{x_1,\ldots,x_v\}, \mathscr{A})$ and a (v',b',r,k)-configuration $(Y = \{y_1,\ldots,y_v\}, \mathscr{B})$, with X and Y being two disjoint sets.

Then we can swap one element in one block of $\mathscr{A}$ for one element in one block of $\mathscr{B}$ to obtain a $(v+v', b+b', r, k)$ configuration. It is trivial to check that this is indeed a configuration. By iterating this method, several configurations can be combined so as to obtain configurations with v and b as large as desired, once r and k are fixed.

The next example illustrates this construction:

$$X = \{x_1,x_2,\ldots,x_4\},$$
$$\mathscr{A} = \{\{x_1,x_2\},\{x_1,x_3\},\{x_1,x_4\},\{x_2,x_3\},\{x_2,x_4\},\{x_3,x_4\}\},$$
$$Y = \{y_1,y_2,\ldots,y_8\},$$
$$\mathscr{B} = \{\{y_1,y_2\},\{y_1,y_3\},\{y_1,y_4\},\{y_2,y_5\},\{y_2,y_6\},\{y_3,y_7\},$$
$$\{y_3,y_8\},\{y_4,y_5\},\{y_4,y_6\},\{y_5,y_7\},\{y_6,y_8\},\{y_7,y_8\}\}.$$

We now can build the following configuration:

$$Z = X \cup Y = \{x_1, \ldots, x_4, y_1, \ldots, y_8\},$$
$$\mathscr{C} = \{\{x_1, x_2\}, \{x_1, x_3\}, \{x_1, x_4\}, \{x_2, \boldsymbol{y_7}\}, \{x_2, x_4\}, \{x_3, x_4\},$$
$$\{y_1, y_2\}, \{y_1, y_3\}, \{y_1, y_4\}, \{y_2, y_5\}, \{y_2, y_6\}, \{y_3, y_7\},$$
$$\{y_3, y_8\}, \{y_4, y_5\}, \{y_4, y_6\}, \{y_5, y_7\}, \{y_6, y_8\}, \{\boldsymbol{x_3}, y_8\}\},$$

where we have swapped x_3 in the block $\{x_2, x_3\}$ of $\mathscr{A}$ for y_7 in the block $\{y_7, y_8\}$ of $\mathscr{B}$ to obtain the new blocks $\{x_2, y_7\}$ and $\{x_3, y_8\}$ of $\mathscr{C}$. The shortcoming of this construction is that in general we cannot swap more than once. Indeed, if in the previous example we swap block elements again, we could end up with something that is not a configuration. Assume we now swap y_7 in block $\{y_5, y_7\}$ of $\mathscr{C}$ for x_1 in block $\{x_1, x_2\}$ of $\mathscr{C}$, to obtain the new blocks $\{y_7, x_2\}$ and $\{y_5, x_1\}$, we get

$$Z = X \cup Y = \{x_1, \ldots, x_4, y_1, \ldots, y_8\},$$
$$\mathscr{D} = \{\{\boldsymbol{y_7}, x_2\}, \{x_1, x_3\}, \{x_1, x_4\}, \{x_2, \boldsymbol{y_7}\}, \{x_2, x_4\}, \{x_3, x_4\},$$
$$\{y_1, y_2\}, \{y_1, y_3\}, \{y_1, y_4\}, \{y_2, y_5\}, \{y_2, y_6\}, \{y_3, y_7\},$$
$$\{y_3, y_8\}, \{y_4, y_5\}, \{y_4, y_6\}, \{y_5, \boldsymbol{x_1}\}, \{y_6, y_8\}, \{\boldsymbol{x_3}, y_8\}\}.$$

The above is not a configuration, because there are two blocks in $\mathscr{D}$ which share both of their points: $\{y_7, x_2\}$ and $\{x_2, y_7\}$.

3.2.2. Combining several copies of the same configuration

If two copies of the same (v, b, r, k)-configuration $(X, \mathscr{A})$ are to be combined then we can swap as many block elements as $vr - 1$ whenever x_i in the first copy is swapped for the corresponding x'_i in the corresponding block in the second copy. In this way we obtain a $(2v, 2b, r, k)$-configuration.

Analogously we can combine n copies of the same configuration and obtain a (nv, nb, r, k)-configuration. Again it is straightforward to prove that what we obtain is a configuration.

The next example illustrates a combination of three copies of the same configuration:

$$X = \{x_1, x_2, x_3, x_4\},$$
$$\mathscr{A} = \{\{x_1, x_2\}, \{x_1, x_3\}, \{x_1, x_4\}, \{x_2, x_3\}, \{x_2, x_4\}, \{x_3, x_4\}\},$$
$$X' = \{x'_1, x'_2, x'_3, x'_4\},$$
$$\mathscr{A}' = \{\{x'_1, x'_2\}, \{x'_1, x'_3\}, \{x'_1, x'_4\}, \{x'_2, x'_3\}, \{x'_2, x'_4\}, \{x'_3, x'_4\}\},$$
$$X'' = \{x''_1, x''_2, x''_3, x''_4\},$$
$$\mathscr{A}'' = \{\{x''_1, x''_2\}, \{x''_1, x''_3\}, \{x''_1, x''_4\}, \{x''_2, x''_3\}, \{x''_2, x''_4\}, \{x''_3, x''_4\}\},$$

with X, X' and X'' being pairwise disjoint sets. The resulting configuration is

$$X \cup X' \cup X'' = \{x_1, x_2, x_3, x_4, x'_1, x'_2, x'_3, x'_4, x''_1, x''_2, x''_3, x''_4\},$$
$$\mathscr{B} = \{\{x_1, x'_2\}, \{x''_1, x_3\}, \{x_1, x_4\}, \{x_2, x''_3\}, \{x'_2, x_4\}, \{x_3, x_4\},$$
$$\{x'_1, x_2\}, \{x'_1, x'_3\}, \{x''_1, x'_4\}, \{x'_2, x'_3\}, \{x_2, x'_4\}, \{x'_3, x''_4\},$$
$$\{x''_1, x''_2\}, \{x_1, x''_3\}, \{x'_1, x''_4\}, \{x''_2, x_3\}, \{x''_2, x''_4\}, \{x''_3, x'_4\}\}.$$

3.2.3. Comparison of the two constructions

A good point of the construction in Section 3.2.1 is that it allows combining two configurations that can be different. However, the resulting configuration connects the two original configurations only weakly, because only two points can be swapped. To illustrate why this is a limitation, consider the application of configurations in the rest of this article: points represent keys and blocks represent users (a user holds the set of keys in her corresponding block). Now, if the two users who have mixed blocks (containing keys from both initial configurations) go off-line, the connection between the sets of users in the two original configurations disappears.

On the other hand, the construction in Section 3.2.2 can only be used to combine copies of the same configuration, but it can combine any number of copies and it allows any number of swaps, so that the resulting configuration can be strongly connected.

4. A peer-to-peer UPIR protocol based on configurations

Consider a peer-to-peer (P2P) community consisting of b users. Assume a dealer who creates a key pool in the following way:

(1) The dealer creates v keys and distributes them into b blocks of size k each according to a (v, b, r, k)-configuration.[2]
(2) The dealer confidentially sends one block of k keys to each user (no two users get the same block). *E.g.*, if each user has got a public–private key pair, confidentiality can be achieved by sending the block of keys encrypted under the user's public key. Let A_i be the block assigned to user u_i, for $i = 1$ to b.

[2] Some flexibility in the choice of parameters is affordable in order to facilitate finding a (v, b, r, k)-configuration. If necessary, b can be somewhat larger than the actual number of users and v, r, k can be chosen appropriately.

(3) The dealer erases the v keys from its storage. If a trusted device such as a smart card is used as a dealer, it can be assumed that keys are forgotten by the dealer after distribution.

Note that Protocols 1 and 2 above use very specific configurations. Protocol 1 uses a $(1, b, b, 1)$-configuration and Protocol 2 a $(b(b-1)/2, b, 2, b-1)$-configuration.

A variant of the above initialization process is to allow the users to send to the dealer their preferences about which other users they would like to share keys with. The dealer could take this input into account to the extent possible when assigning blocks of keys to users.

At the end of the process, by Lemma 1 the block of keys of each user intersects $k(r-1)$ other users' blocks. Consider now a storage pool consisting of v memory sectors, each corresponding to one key in the key pool. A protocol for peer-to-peer UPIR among the b users is specified next.

Protocol 3 (Configuration-based P2P UPIR)(q_i)

(1) In order to submit a query q_i to a database or search engine, user u_i randomly selects one of the k keys in her block. Let x_{ij} be the selected key and $U_j^i = \{u_{j1}^i, \ldots, u_{j(r-1)}^i\}$ be the set of $r-1$ users with whom u_i shares x_{ij} according to the configuration used for key distribution. (Note that the sets $U_1^i, \cdots, U_k^i$ are disjoint due to the configuration structure.)
(2) u_i reads the memory sector m_{ij} corresponding to key x_{ij} and decrypts it under x_{ij}. Five cases can arise depending on the outcome of decryption:
   (a) The outcome is garbage. In this case, u_i encrypts q_i using a symmetric cipher keyed by x_{ij} and records the encrypted query in sector m_{ij}.
   (b) The outcome is a query q_j issued by some user in U_j^i, who expects some other user in U_j^i to submit it on her behalf. In this case, u_i submits q_j to the database/search engine and records in sector m_{ij} the answer obtained after encrypting it under key x_{ij}. Thereafter, u_i goes back to Step 1 to select a new key and obtain assistance in submitting q_i to the database/search engine from someone in the group of $r-1$ users sharing the new key with u_i.
   (c) The outcome is the answer to a previous query q_j' issued by some user in U_j^i and previously submitted by some other user in U_j^i to the database/search engine on behalf of that user. Since this answer has not yet been read by the user in U_j^i (a user is assumed to erase the query answer when she reads it), u_i goes back to Step 1 to select a new key and obtain assistance in the submission of her own query q_i.
   (d) The outcome is a query q_i' previously issued by u_i, who expects some user in U_j^i to submit it on u_i's behalf. Since there is a previous query pending to be serviced by some user in U_j^i, u_i goes back to Step 1 to select a new key and obtain assistance with the submission of her new query q_i.
   (e) The outcome is the answer to a previous query q_i' issued by u_i and previously submitted by some user in U_j^i to the database/search engine on behalf of u_i. In this case, u_i reads the answer, then encrypts her new query under key x_{ij} and finally records the encrypted query in sector m_{ij}.

It can be seen that Protocol 3 will iterate until u_i can have her query submitted to the database/search engine by some other user. Similarly to what happened in Protocol 2, jamming at Step 2c above can be thwarted by imposing a timeout on how long a shared memory sector must hold a certain query answer before the latter can be overwritten. Jamming at Step 2d is prevented by requiring all users to check their shared memory sectors as discussed below.[3]

Like for Protocols 1 and 2 above, once user u_i has managed to submit her query q_i, u_i must keep frequently calling the protocol with a garbage query and fixed selected user u_j until she can collect the answer to q_i.

If a user does not have queries to submit and never runs Protocol 3, she does not contact the database; if the number of users contacting the database is very small (*e.g.* only two) there are problems: (i) the database may infer who is submitting what query, and (ii) the delay until a query answer can be collected can be too long. To prevent this, we require that all users u_i do the following at regular time intervals: scan in a random order the memory sectors m_{ij} shared with other users u_j until either all sectors have been read or a sector containing a query is found, in which case the query is submitted and the corresponding answer is recorded in that sector.

5. Performance and privacy

We examine in this section the influence of the configuration parameters k and r on performance and privacy. The other two parameters do not need discussion: b is the (fixed) number of users in the P2P community and v is the number of keys and depends on k, r and b according to Eq. (1).

[3] The probability of jamming at Step 2c is mitigated by choosing a configuration with a large r (*i.e.* each memory sector m_{ij} is shared by a large set U_j^i); by Eq. (1), for a fixed number of users b and number k of keys given to each user, increasing r can be done by decreasing the number of keys v (thus tending to Protocol 2); for fixed b and v, increasing r implies increasing k.

5.1. Performance

First we deal with performance in terms of required keys and required storage. The proposed set-up process based on a (v, b, r, k)-configuration is compared with the situation of Protocol 2 in which every user shares a different key with every other user (complete connection graph). It turns out that performance improvement is controlled by parameter r.

Lemma 2. *If $r > 2$ it holds that:*

- *the number of keys and memory sectors required using a (v, b, r, k)-configuration is less than the number of keys and memory sectors required in the case of a complete graph;*
- *the overall number of keys stored by the users with a (v, b, r, k)-configuration is less than in the case of a complete graph.*

Proof. With a complete graph among the b users, the number of required keys and memory sectors is $b(b-1)/2$. Each user stores $b-1$ keys, so that the overall number of keys stored by the users is $b(b-1)$.

With configurations, the number of required keys and memory sectors is $v = bk/r$ (Eq. (1)). The overall number of keys stored by the users is bk. Now, from Lemma 1 it follows that $k(r-1) \leqslant b-1$ (the number of blocks intersecting a specific block cannot be greater than $b-1$); thus

$$\frac{bk}{r} \leqslant \frac{b(b-1)}{r(r-1)}.$$

So for $r > 2$ there is a reduction in the number of required keys and memory sectors with respect to the complete graph case. Similarly, since $bk \leqslant b(b-1)/(r-1)$, for $r > 2$ there is a reduction in the overall number of keys stored by the users. □

In addition to storage, another performance metric is how long does it take for u_i to get her query submitted and answered. Clearly, the greater the number r with whom u_i shares the selected key x_{ij}, the shorter is the expected waiting time.

Therefore, performance improves as r increases.

5.2. Privacy

If a good symmetric cipher is used for encryption, the encrypted contents stored in any memory sector are indistinguishable from garbage (see [20] for a review of the properties of the output of a symmetric cipher). Thus, to an external intruder not in $\{u_i\} \cup U_j^i$ the content of sector m_{ij} is indistinguishable from garbage; therefore, such an intruder does not gain any information on the queries submitted nor the query answers received by users in $\{u_i\} \cup U_j^i$. As to leakage possibilities of x_{ij} to external intruders, they increase with the size $r-1$ of U_j^i. Therefore, privacy against external intruders degrades as r increases.

Within U_j^i, the $r-1$ users do not know in principle to which other user in $\{u_i\} \cup U_j^i$ do the queries and query answers correspond. From this remark and those in the performance section above, one might be tempted to take r as large as possible, that is, a single key shared by all b users ($r = b$ and $k = v = 1$), which yields Protocol 1. However, we have argued that Protocol 1 does not provide very good privacy in front of other users nor external intruders.

It seems better for u_i to limit (pseudonymous) visibility of her query and its answer to those parties strictly needed: the database/search engine and a set of users just large enough so that the expected waiting time to get the query answer is not too long. Indeed, if u_i can select x_{ij} among $k > 1$ different keys at Step 1 of Protocol 3, where each key is shared by a disjoint set of users (see proof of Lemma 1), users in U_j^i only see on average 1 out of k queries issued by u_i (and 1 out of k query answers received by u_i). Therefore, the risk that a user in U_j^i can profile and thereby re-identify u_i decreases as k increases.

Finally, let us examine the privacy of user u_i in front of the database or search engine. The queries issued by u_i are submitted by the $k(r-1)$ users with whom u_i shares keys. In fact, each u_j in that group of $k(r-1)$ users submits on average a fraction $1/(k(r-1))$ of the queries issued by u_i. But u_j may also submit other queries corresponding to other users different from u_i with whom u_j shares a key. Therefore the query profile of u_i is diffused among the $k(r-1)$ users with whom u_i shares a key and confused among the other queries submitted by those users.

In summary, the greater r, the better is performance; the smaller r, the better is privacy in front of external intruders; the greater k, the better is privacy in front of the other users; the greater $k(r-1)$, the greater is privacy in front of the database/search engine. From Lemma 1, it follows that $k(r-1) \leqslant b-1$, so the optimal situation is $k(r-1) = b-1$; the construction [25] yields configurations which are optimal in that sense.

6. Simulation results

After showing the potential of configurations to reduce the amount of cryptographic material while preserving a good privacy level in front of the database/search engine and the rest of users, it remains to evaluate the impact of Protocol 3 on the response time, that is, how long does it take since a user submits a query using the protocol until the user gets the query answer. The protocol should not introduce too much overhead, that is, the overall response time using Protocol

3 should not be much longer than the response time incurred when directly submitting the query to the search engine/database (with no UPIR).

In order to assess this overhead, we have carried out two-hour simulations with a realistic parameter choice:

- For a first small simulation we took the (34,34,6,6)-configuration listed in [13], that is, a configuration consisting of 34 peer users, 34 keys and with each user sharing 6 keys with 6 different users;
- In order to test the protocol for a large community of peers we considered a (993,993,32,32) configuration obtained using the construction in [25]. In this case the system consists of 993 peer users, 993 keys and with each user sharing 32 keys with 32 different users.
- The query behavior of users has been modeled by assuming that the time between two successive queries by the same user is a random variable following an exponential distribution with parameter λ, which gives an expected time $1/\lambda$ between queries.
- Different load conditions have been tried by taking several values for the expected time $1/\lambda$ between successive queries: 1200, 900, 600, 300, 120, 90, 60 and 30 s.
- For each time between queries, two options have been tried:

Table 1
For 34 peer users, average response time overhead as a function of the expected time between successive queries by a user when users do nothing unless they have a query to submit (no regular scanning of shared memory sectors).

1200	900	600	300	120	90	60	30
1165.20	872.97	615.26	355.79	156.44	115.06	77.28	44.03

Table 2
For 34 peer users, average response time overhead as a function of the expected time between successive queries by a user and the time interval for regular scanning of shared memory sectors by every user.

Time btw. queries	Scan interval						
	15	30	45	60	120	240	300
1200	5.95	8.63	11.26	15.28	26.54	55.22	71.34
900	5.86	8.38	11.34	14.43	26.36	53.98	67.72
600	5.84	8.30	11.02	13.74	25.71	55.16	72.68
300	5.92	8.43	11.36	14.29	28.08	67.07	90.01
120	5.97	8.79	12.04	16.34	37.86	80.30	96.72
90	6.02	9.01	12.46	17.75	42.64	74.47	83.54
60	6.17	9.67	14.46	20.44	43.82	59.96	66.09
30	6.65	11.71	18.36	23.61	31.34	36.85	38.49

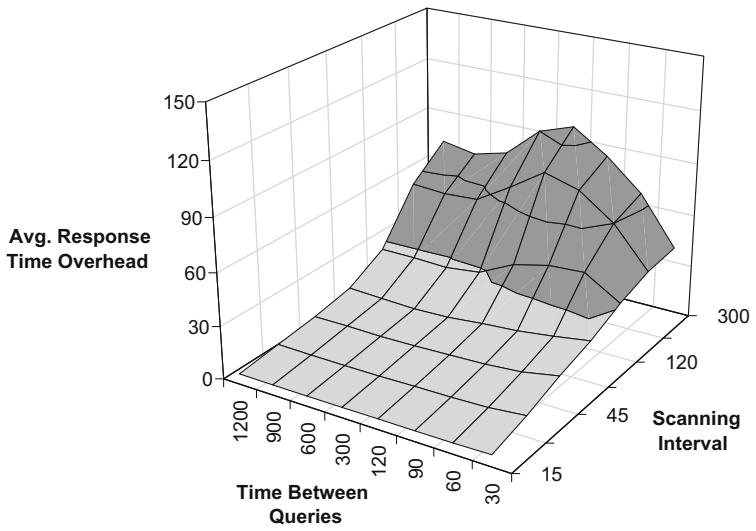

Fig. 1. For 34 peer users, average response time overhead as a function of the expected time between successive queries by a user and the time interval for regular scanning of shared memory sectors by every user.

- Users do nothing unless they have a query to submit;
- All users do as described in the last paragraph of Section 4, that is, they scan and process the memory sectors shared with other users at regular time intervals. Regular intervals of 15, 30, 45, 60, 120, 240 and 300 s have been considered.

- For each parameter combination, 10 simulations have been conducted and the average time overhead added by Protocol 3 to the query response time has been computed (that is, the time increase with respect to direct query submission without UPIR). The results for the smaller configuration can be seen in Tables 1 and 2, and Fig. 1. On the other hand, the results for the larger configuration are shown in Tables 3 and 4, and Fig. 2.

One can see that the (993,993,32,32)-configuration yields a shorter response time overhead than the (34,34,6,6)-configuration. The reason is that the larger configuration is optimal in the sense described at the end of Section 5.2, that is, it satisfies $k(r-1) = 32 * 31 = 992 = b - 1$; hence, with the larger configuration a user has more chances to get her queries submitted by other users.

Table 3
For 993 users, average response time overhead as a function of the expected time between successive queries by a user when users do nothing unless they have a query to submit (no regular scanning of shared memory sectors).

1200	900	600	300	120	90	60	30
963.19	811.81	563.44	295.43	124.40	95.33	64.17	33.51

Table 4
For 993 users, average response time overhead as a function of the expected time between successive queries by a user and the time interval for regular scanning of shared memory sectors by every user.

Time btw. queries	Scan interval						
	15	30	45	60	120	240	300
1200	5.13	5.40	5.98	6.92	11.40	21.47	26.96
900	5.13	5.35	5.90	6.59	10.59	20.25	25.94
600	5.18	5.28	5.66	6.26	9.75	18.90	24.37
300	5.12	5.22	5.53	6.04	9.25	20.75	32.75
120	5.13	5.21	5.57	6.20	13.8	65.00	78.60
90	5.13	5.22	5.68	6.57	27.42	61.70	68.55
60	5.14	5.27	6.07	8.50	34.74	50.12	52.70
30	5.14	6.29	13.72	18.75	26.25	30.28	30.74

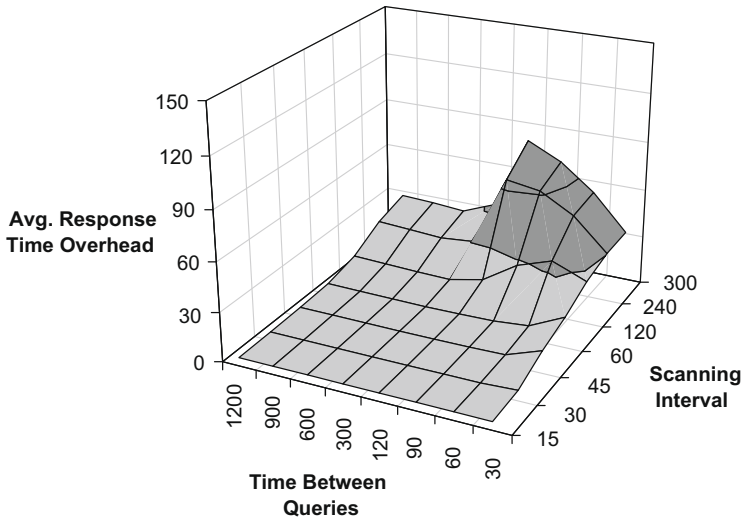

Fig. 2. For 993 users, average response time overhead as a function of the expected time between successive queries by a user and the time interval for regular scanning of shared memory sectors by every user.

As it can be seen in Tables 1 and 3, if users do nothing unless they have a query of their own to submit, the shorter the expected time between two successive queries by the same user, the shorter the average response time overhead. This is explained by the design of Protocol 3, in which users must take care of other users' queries before submitting their own. However, the reported response time overheads are much too long to be acceptable. This justifies the requirement that every user should scan her shared memory sectors at regular intervals and submit the first query from some other user found in those sectors.

Tables 2 and 4 and Figs. 1 and 2 show the average response time overhead when regular scanning is implemented. It can be seen that, even with a rather long scanning interval, an acceptable overhead is obtained. Note that, when scanning intervals are very short, the overhead increases slightly as the time between queries decreases: the reason is that, with a short scanning interval and a short time between queries, the shared memory sectors get filled with uncollected query answers, so that submitting a new query becomes increasingly difficult.

If we take 30 s as the cut-off value for an acceptable overhead, the region of acceptable overhead is depicted in light grey in Figs. 1 and 2.

A possible improvement would be to dynamically update the scanning interval as a function of the time between user queries and the current overhead response time. The aim of the updating strategy would be to determine the scanning interval that is needed to guarantee that the overhead stays below a target value.

7. A dealer-free extension of the protocol

In the peer-to-peer UPIR protocol proposed in Section 4, a dealer is required to confidentially distribute secret keys to each group member. In some cases, it may be difficult to find such a trusted dealer in a peer-to-peer scenario. Hence, the practicality of the protocol can be improved if we can remove the trusted dealer in the proposed UPIR protocol. We begin with some cryptographic notions that will be useful for removing the dealer in the proposed UPIR protocol.

7.1. Burmester–Desmedt group key agreement

A group key agreement (GKA) protocol is a cryptographic primitive allowing two or more members to establish a common secret via open networks. After execution of a GKA protocol, only the group members can compute the shared secret key. Attackers can obtain no information about the shared key by monitoring the communication of the group members. Many efficient GKA protocols have been proposed and, among them, the Burmester–Desmedt protocol [3] is the most efficient one in communication and computation for more than three group members.

The Burmester–Desmedt protocol is a two-round n-party protocol. It assumes a finite cyclic group[4] generated by a generator g of prime order p. Here, g and p are system parameters. In the first round, each user $\mathcal{U}_i$ chooses a random $\delta_i \in \mathbb{Z}_p^*$ and broadcasts $z_i = g^{\delta_i}$. In the second round, each user $\mathcal{U}_i$ broadcasts $X_i = (z_{i+1}/z_{i-1})^{\delta_i}$. Finally, any user $\mathcal{U}_i$ can compute the common secret key

$$K = z_{i-1}^{n\delta_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \cdots X_{i-2} = g^{\delta_1\delta_2+\delta_2\delta_3+\cdots+\delta_n\delta_1},$$

where the subscripts are computed modulo n. The Burmester–Desmedt protocol was proven secure in 2003 [15]. That is, given that an attacker can only monitor the communication among the group members, the attacker cannot distinguish the shared key K from a random element in the finite cyclic group generated by g.

7.2. Linkable ring signatures

Ring signatures introduced in [23] can achieve anonymity for ad hoc groups *without* any trusted manager. They are used to convince any third party that *at least* one member in an ad hoc group has indeed issued the signature on behalf of the group. Ring signatures are very suitable for anonymity applications in P2P scenarios due to their attractive properties. Ring signatures are *set-up free*. Ring signatures require no managers to initialize the system. All signers publish their public keys to form a public-key list and any player wishing to generate a ring signature later appends her own public key to the list and can generate a valid ring signature. Ring signatures are *cooperation-free*. This refers to the capability of having a ring member produce a ring signature for any message independently. Hence, a ring signature requires no interactions or cooperations among ring members provided that all the members' public keys are known. Ring signatures can guarantee *identity privacy*. This means that the signer is anonymous and no one can identify the author of a given ring signature. Since there is no trusted manager in ring signatures, the anonymity of ring signatures is perfect and cannot be revoked.

Linkable ring signatures [19] allow anyone to determine whether two signatures are signed by the same anonymous member (*linkability* property). If a user signs only once on behalf of a group, the user still enjoys anonymity similar to the one offered by conventional ring signatures.

[4] In this section, the term "group" may mean an algebraic system or a set of users. As the precise meaning is clear from the context, no further attention will be paid to this distinction.

7.3. Basic ideas

After the set-up process previous to Protocol 3 and due to the (v, b, r, k)-configuration structure, the group members have v keys where each key is shared by r users and each user has k keys. As assumed throughout this article, users may not know each other's identity even when they share a key, that is, users can share keys with identity privacy. The expected features of our extension can be summarized as follows:

- The set-up process should not require a dealer.
- The keys should be shared by the group members according to a (v, b, r, k)-configuration structure.
- The group members sharing any key are anonymous.

Before going into details, let us sketch the basic ideas of our dealer-free extension. We want the group members to jointly simulate the dealer in a distributive manner. Note that each block of the configuration corresponds to a user and each key is shared by a sub-group consisting of r users. Hence we can let each sub-group run a group key agreement protocol to share a common secret key following the (v, b, r, k)-configuration structure. For identity privacy, we exploit linkable ring signatures to achieve anonymity.

7.4. Extended set-up process without a dealer

Consider the same scenario as at the beginning of Section 4. Assume a P2P community consisting of b users who agree on the parameters of a (v, b, r, k)-configuration. Each user has a registered public key, for instance, a PKI-based public key, known by other users. Also, assume that all users share a linkable ring signature scheme, for instance, the scheme in [19]. The b users create a key pool in the following way without a dealer:

(1) Each user generates a linkable ring signature and sends it to other users. Then each user holds b linkable ring signatures. Each signature corresponds to an anonymous member of the b users. Let these ring signatures be ordered in some order. After ordering them, denote these signatures by $1, 2, \ldots, b$, where each number corresponds to an anonymous user. Hence, without loss of generality, we simply use i for $i = 1, \ldots, b$ to represent a certain anonymous user.
(2) Each user runs the greedy algorithm or one of the constructions in Section 3 to find a (v, b, r, k)-configuration. If none can be found, then the users should agree on a different set of parameters until a configuration can be found for those parameters. Note that the users generate the same configuration, because they run exactly the same procedures. The b blocks are viewed as b users and the ith block corresponds to the anonymous user encoded as i. Each user is in k sub-groups and each sub-group consists of r users. There are in total v sub-groups but different sub-groups may intersect.
(3) For each sub-group, the users in the sub-group run the Burmester–Desmedt group key agreement protocol to share a common secret key. Messages produced during the protocol execution are all signed by the users with the shared linkable ring signature scheme. Let $\mathbb{G}$ be a finite cyclic group generated by a generator g of prime order p. For a sub-group consisting of users $i_1, \ldots, i_r$, they establish a shared secret key as follows:
   - In the first round, for $j = 1, \ldots, r$, each user i_j chooses a random $\delta_j \in \mathbb{Z}_p^*$ and broadcasts to the sub-group $z_j = g^{\delta_j}$ and a linkable ring signature of z_j using the shared linkable ring signature scheme. Then any user can verify whether z_j comes from the anonymous user i_j due to the linkability of the underlying linkable ring signature scheme. If all verifications of the users pass, the users enter the second round of the group key agreement protocol.
   - In the second round, each user i_j broadcasts $X_j = (z_{j+1}/z_{j-1})^{\delta_j}$ and a linkable ring signature of X_j using the shared linkable ring signature scheme. Then any user can verify whether X_j comes from the anonymous user i_j due to the linkability of the underlying linkable ring signature scheme. If all verifications of the users pass, the users of the sub-group proceed to the final step below.
   - Finally, any user i_j can compute the common secret key $K = z_{j-1}^{r\delta_j} \cdot X_j^{r-1} \cdot X_{j+1}^{r-2} \cdots X_{j-2} = g^{\delta_1\delta_2 + \delta_2\delta_3 + \cdots + \delta_r\delta_1}$, where the subscripts are computed modulo r.

After the above initialization process, each sub-group consisting of r users have a common secret key. Each user has k secret keys as each user is in k sub-groups. Since there are totally v sub-groups, v secret keys are established and each key is shared by r users. Also, the users do not know the identities of other users sharing the same secret key in their sub-group due to the anonymity of the underlying ring signatures. Hence, the above set-up process without a dealer perfectly simulates the dealer-based initialization process in Section 4.

7.5. Complexity assessment

We next analyze the complexity of the above dealer-free extension. In Section 6, we have given figures for the response time of the protocol with a trusted dealer; thus, to avoid repetition, we just investigate here the additional time delay introduced by the cryptographic operations for each user due to the lack of a dealer.

In Step 1 of the set-up process described in Section 7.4 (form a peer-to-peer group), the additional operation for each user is to generate one linkable ring signature and verify $(b - 1)$ signatures (each user holds b linkable ring signatures but one of

them is the user's own signature) from other $b-1$ users. According to [19], generating one ring signature requires $2b+1$ modular (multi-base) exponentiations (we do not differentiate a multi-base exponentiation from a single-base exponentiation as both have a similar computation cost) and b hash operations. For verification, each linkable ring signature requires $2b$ exponentiations and b hash operations. Hence, the total additional cost in that Step 1 is $2b(b-1)+2b+1=2b^2+1$ exponentiations and b^2 hashes. Since, for existing cryptographic hash functions in use, the computation cost of a hash operation is much less than that of an exponentiation, the main additional overhead is $2b^2+1$ exponentiations.

Step 2 of the set-up process described in Section 7.4 does not count as additional delay, because a configuration is needed anyway even if a dealer is used.

Finally, in Step 3 of the set-up process the main additional cost is $2\times(2r^2+1)$ exponentiations to generate two ring signatures and verify $2(r-1)$ ones, plus 2 more exponentiations needed by the Burmester–Desmedt protocol [3].

According to the measurement of the cryptographic library MIRACL [21] in a PC environment, the time to compute an exponentiation is about 0.1ms. At Step 1, for a middle-sized group of $b=100$ users, the additional delay incurred is about two seconds. At Step 3, if one takes $r<50$ (r is typically much smaller than b), the additional delay incurred is less than one second. Hence, the overhead caused by doing without a trusted dealer is affordable in practice.

8. Conclusion

Relaxing PIR seems a pragmatic approach to obtain working systems offering some degree of query privacy against uncooperative search engines or databases. Practical relaxations proposed in the literature are standalone and rely on a single user cloaking her queries by either submitting them masked with fake keywords with similar frequency or submitting them unaltered but camouflaged in a cloud of ghost queries. We have introduced an alternative relaxation named user-private information retrieval, in which the user query history rather than the query is cloaked; indeed, the user seeks assistance from a P2P community who submit queries on her behalf. In this way, any query can be submitted without caring for keyword frequencies nor swamping search engines with ghost queries. The level of privacy achieved is proportional to the connectivity $k(r-1)$ of the P2P community. Furthermore, we have given simulation results which show the practicality of the proposed protocol and we have described a protocol extension where no dealer is needed.

From the combinatorial point of view, we have also contributed some constructions of configurations (the structure used for key and storage management).

References

[1] AOL Search Data Scandal, August 2006. <http://en.wikipedia.org/wiki/AOL_search_data_scandal>.
[2] A. Beimel, Y. Ishai, T. Malkin, Reducing the servers' computation in private information retrieval: PIR with preprocessing, Journal of Cryptology 17 (2004) 125–151.
[3] M. Burmester, Y. Desmedt, A secure and efficient conference key distribution system, in: Eurocrypt'94, LNCS 950, Springer-Verlag, 1994, pp. 275–286.
[4] B. Chor, N. Gilboa, M. Naor, Private information retrieval by keywords, Technical Report TR CS0917, Department of Computer Science, Technion, 1997.
[5] B. Chor, O. Goldreich, E. Kushilevitz, M. Sudan, Private information retrieval, in: IEEE Symposium on Foundations of Computer Science (FOCS), 1995, pp. 41–50.
[6] B. Chor, O. Goldreich, E. Kushilevitz, M. Sudan, Private information retrieval, Journal of the ACM 45 (1998) 965–981.
[7] J. Domingo-Ferrer, M. Bras-Amorós, Peer-to-peer private information retrieval, in: Privacy in Statistical Databases-PSD 2008, LNCS 5262, Springer-Verlag, 2008, pp. 315–323.
[8] J. Domingo-Ferrer, A. Solanas, J. Castellà-Roca. $h(k)$-Private information retrieval from privacy-uncooperative queryable databases, Online Information Review, 3 (4) (2009). Goopir software downloadable from: <http://unescoprivacychair.urv.cat/goopir>.
[9] S. Goldwasser, S. Micali, Probabilistic encryption, Journal of Computer and Systems Science 28 (1) (1984) 270–299.
[10] H. Gropp, On the history of configurations, in: International Symposium on Structures in Mathematical Theories, Bilbo, 1990, Euskal Herriko Unibertsitatea, pp. 263–268
[11] H. Gropp, Configurations between geometry and combinatorics, Discrete Applied Mathematics 138 (1-2) (2004) 79–88. Optimal discrete structures and algorithms (ODSA 2000).
[12] H. Gropp, Existence and enumeration of configurations, Bayreuther Mathematische Schriften 74 (2005) 123–129.
[13] H. Gropp, Configurations, in: C.J. Colbourn, J.H. Dinitz (Eds.), Handbook of Combinatorial Designs, Chapman & Hall/CRC, Boca Raton, FL, 2007, pp. 353–355.
[14] D.C. Howe, H. Nissenbaum, TrackMeNot: resisting surveillance in web search, in: I. Kerr, C. Lucock, V. Steeves (Eds.), Lessons from the Identity Trail: Privacy, Anonymity and Identity in a Networked Society, Oxford University Press, Oxford UK, 2009, pp. 409–428. Software downloadable from: http://www.mrl.nyu.edu/~dhowe/trackmenot/.
[15] J. Katz, M. Yung, Scalable protocols for authenticated group key exchange, in: Crypto'03, LNCS 2729, Springer-Verlag, 2003, pp. 110–125.
[16] E. Kushilevitz, R. Ostrovsky, Replication is not needed: single database, computationally-private information retrieval, in: Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, 1997, pp. 364–373.
[17] J. Lane, P. Heus, T. Mulcahy, Data access in a cyber world: making use of cyberinfrastructure, Transactions on Data Privacy 1 (1) (2008) 2–16.
[18] J. Lee, D.R. Stinson, A combinatorial approach to key predistribution for distributed sensor networks, in: Wireless Communications and Networking Conference-WCNC 2005, vol. 2, 2005, pp. 1200–1205.
[19] J.K. Liu, V.K. Wei, D.S. Wong, Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract), in: ACISP'04, LNCS 3108, Springer-Verlag, 2004, pp. 325–335.
[20] A.J. Menezes, P.C. van Oorschot, S.A. Vanstone (Eds.), Handbook of Applied Cryptography, CRC Press, Boca Raton, 1997.
[21] Multiprecision Integer and Rational Arithmetic C/C++ Library (MIR-ACL). <http://www.shamus.ie/>.
[22] R. Ostrovsky, W.E. Skeith III, A survey of single-database PIR: techniques and applications, in: Public Key Cryptography-PKC 2007, Lecture Notes in Computer Science, vol. 4450, Berlin Heidelberg, 2007, pp. 393–411.
[23] R.L. Rivest, A. Shamir, Y. Tauman, How to leak a secret, in: Proceedings of the Asiacrypt'01, LNCS 2248, Springer-Verlag, 2001, pp. 552–565.
[24] D.R. Stinson, Combinatorial Designs: Constructions and Analysis, Springer-Verlag, New York, 2003.
[25] K. Stokes, M. Bras-Amorós, Optimal configurations for peer-to-peer private information retrieval, manuscript, 2008.
[26] A.S. Tanenbaum, Computer Networks, fourth ed., Prentice Hall, Upper Saddle River, NJ, 2002.
[27] The Tor Project, Inc. "Tor: Overview". <http://torproject.org/overview.html.en>.

Josep Domingo-Ferrer is a Full Professor of Computer Science and an ICREA-Acadèmia Researcher at Universitat Rovira i Virgili, Tarragona, Catalonia, where he holds the UNESCO Chair in Data Privacy. He received with honors his M.Sc. and Ph.D. degrees in Computer Science from the Universitat Autònoma de Barcelona in 1988 and 1991, respectively (Outstanding Graduation Award). He also holds a M.Sc. in Mathematics. His fields of activity are data privacy, data security and cryptographic protocols. He has received three research awards and four entrepreneurship awards, among which the ICREA Acadèmia Research Prize from the Government of Catalonia. He has authored 3 patents and over 220 publications, one of which became an ISI highly-cited paper in early 2005. He has been the co-ordinator of EU FP5 project CO-ORTHOGONAL and of several Spanish funded and US funded research projects. He currently co-ordinates the CONSOLIDER "ARES" team on security and privacy, one of Spain's 34 strongest research teams. He has chaired or co-chaired nine international conferences and has served in the program committee of over 70 conferences on privacy and security. He is a co-Editor-in-Chief of "Transactions on Data Privacy" and an Associate Editor of three international journals. In 2004, he was a Visiting Fellow at Princeton University.

Maria Bras-Amorós is a Tenured Associate Professor at Universitat Rovira i Virgili in Tarragona. She received the Ph.D. degree in applied mathematics from the Universitat Politècnica de Catalunya in 2003. Part of her doctoral and postdoctoral work was developed at San Diego State University, California. She has been with the Universitat Politècnica de Catalunya and the Universitat Autònoma de Barcelona, and is currently with the Universitat Rovira i Virgili in Tarragona. Her main research interests are in the area of coding theory and discrete mathematics.

Qianhong Wu is a senior researcher with the UNESCO Chair in Data Privacy, Department of Computer Science and Mathematics, Universitat Rovira i Virgili (URV), Tarragona, Catalonia. He received his M.Sc. in Applied Mathematics from Sichuan University in 2001. He got his Ph.D. degree in Cryptography from Xidian University in 2004. He has been a postdoctoral researcher with Wollongong University in Australia and Wuhan University in China. His research interests include public key cryptography, e-commerce security, security and privacy in networks, and private information retrieval. He has been a principal investigator or co-investigator of several Chinese-funded and Australian-funded projects. He has coauthored over 40 publications and has been a reviewer for several international journals.

Jesús Manjón is a computer engineer with the UNESCO Chair in Data privacy and the CRISES Research Group in the Department of Computer Science and Maths at Universitat Rovira i Virgili of Tarragona, Catalonia. He got his M.Sc. in Computer Engineering in 2004 and an M.Sc. in Computer Security in 2008.