J. Cryptol. (2010) 23: 344–371
DOI: 10.1007/s00145-008-9032-z

Journal of
CRYPTOLOGY

How Should We Solve Search Problems Privately?*

Amos Beimel
Dept. of Computer Science, Ben-Gurion University,Be'er Sheva, Israel
beimel@cs.bgu.ac.il

Tal Malkin†
Dept. of Computer Science, Columbia University, New York, NY, USA
tal@cs.columbia.edu

Kobbi Nissim‡
Dept. of Computer Science, Ben-Gurion University, Be'er Sheva, Israel
kobbi@cs.bgu.ac.il

Enav Weinreb
CWI, Amsterdam, The Netherlands
e.n.weinreb@cwi.nl

Communicated by Matthew Franklin

Received 7 December 2007 and revised 2 October 2008
Online publication 6 November 2008

Abstract. Secure multiparty computation allows a group of distrusting parties to jointly compute a (possibly randomized) *function* of their inputs. However, it is often the case that the parties executing a computation try to solve a *search problem*, where one input may have a multitude of correct answers—such as when the parties compute a shortest path in a graph or find a solution of a set of linear equations.

The algorithm for arbitrarily picking one output from the solution set has significant implications on the privacy of the computation. A minimal privacy requirement was put forward by Beimel et al. [STOC 2006] with focus on proving impossibility results. Their definition, however, guarantees a very weak notion of privacy, which is probably insufficient for most applications.

In this work we aim for stronger definitions of privacy for search problems that provide reasonable privacy. We give two alternative definitions and discuss their privacy guarantees. We also supply algorithmic machinery for designing such protocols for a broad selection of search problems.

Key words. Secure computation, Search problems, Privacy, Resemblance

* A preliminary version of this work appeared in A. Menezes, editor, *Advances in Cryptology—CRYPTO 2007*, *Lecture Notes in Computer Science*, vol. 4622, pp. 31–49. Springer, 2007.

† Research partially supported by the NSF (grant No. CCF-0347839).

‡ Research partially supported by the Israel Science Foundation (grant No. 860/06).

1. Introduction

Secure multiparty computation addresses a setting where several distrusting parties want to jointly compute a function $f(x_1, \dots, x_n)$ of their private inputs $x_1, \dots, x_n$ while maintaining the privacy of their inputs. One of the most fundamental, and by now well known, achievements in cryptography (initiated by [3,9,15,21], and continued by a long line of research) shows that, for any feasible function f, there exist secure multiparty protocols for f in a variety of settings.

However, in many cases, what the parties wish to compute is not a function with just a single possible output for each input, and not even a randomized function with a well-defined output distribution. Rather, in many cases the parties are solving a problem where several correct answers (or *solutions*) may exist for a single joint input. For example, the parties may jointly hold a graph and wish to compute a shortest path between two of its vertices or to find a minimal vertex cover in it.[1] Such problems are called *search problems*.

When one tries to apply the known results of secure multiparty computation to search problems, one has first to decide upon a polynomial-time computable *function* that solves the search problem. That is, one should present a specific algorithm that assigns a single output, or a well-defined distribution on the output space, to every possible input.

An approach often taken by designers of secure multiparty protocols for such applications is to arbitrarily choose one of the existing algorithms/heuristics for the search problem and implement a secure protocol for it. This indeed amounts to choosing an arbitrary (possibly randomized) *function* that provides a solution and implementing it securely. The privacy implications of such choices have not been analyzed. It is usually easy to come up with "bad" algorithms that encode in their output choices crucial information about one or more of their inputs, and it may be that algorithms accidentally leak such information even if not maliciously designed. It is clear that if the computed function leaks information on the parties' private inputs, any protocol realizing it, no matter how secure, will also leak this information. Thus, some privacy requirements should be imposed on the chosen input-output functionality.

To illustrate the necessity of a rigid discussion of secure computation of search problems, consider the following setting. A server holds a database with valuable information, and a client makes queries to this database such that there may be many different answers to a single query. The server is interested in choosing the answers to the client's queries in a way that reveals the "least information possible about the database."

The answering strategy the server chooses influences the revealed information. For example, consider a case where the client queries for the name of a person whose details are in the database and satisfy some condition. If the server answers such queries with the details of the appropriate person whose name is the lexicographically first in the database, then it reveals the fact that every person prior to that person in the lexicographic order does not satisfy the given condition.

[1] Another example is when the parties compute an approximation of a function $f()$ as in [11,17,18]. Again, there is potentially more than one correct answer for an instance.

1.1. *This Work*

In this paper we study some of the privacy implications of how the output is chosen for search problems, we propose two privacy requirements and provide constructions achieving them for several problems. Our work generalizes the approach taken in [1], where the problem of private search was introduced in the context of approximation algorithms. Finally, we give examples for problems that admit private search algorithms of certain kinds and do not admit other.

Our work is closest in spirit to that of Beimel et al. [1]. They have put forward what seems to be a minimal requirement of privacy.[2] Informally:

> **Private Search:** *If two instances x, y have an identical set of possible solutions, their outputs should not be distinguished.*

That is, in order for an algorithm to be private, on any input instance its output must only depend on the solution set for that input—not on the *specific* input. In the context of private approximations of functions [11] (and, similarly, for those instances of search problems for which a *unique* solution exists), this turns out to be a significant privacy guarantee—it implies that the approximation algorithm does not leak any information beyond the exact $f(x)$. However, in the context of search problems the implication is potentially much weaker—no information beyond the *entire solution set* of x is leaked. Arguably, for many applications requiring privacy, leaking information up to the entire solution set does not provide a sufficient privacy guarantee. We note that this weak definition was reasonable in the context of [1] because they provide mostly negative results (so a weaker definition corresponds to stronger infeasibility results).

Note that even with the minimal definition of privacy of [1], the notion of private search has so far proved to be very problematic. Search versions of many NP-complete problems do not admit even much weakened notions of private approximation algorithms, and private search is infeasible even for some problems that do admit (non-private) polynomial time search algorithms [1,2].

We are thus faced with a double challenge: first, strengthen the privacy definition, imposing further requirements on the function in order to provide reasonable privacy guarantees. Second, provide protocols implementing the stronger definitions for as wide as possible a class of search problems. This is the goal we tackle in this work.

Which further requirements should be imposed on the outcome of private search algorithms? While the answer to this question may be application dependent, we identify two (incomparable) requirements and study which problems admit those requirements and which techniques can be used to achieve them.

Before elaborating on this, let us start with two naïve proposals—to demonstrate essential privacy considerations arising for search problems and to facilitate our actual proposed definitions and algorithms. In the following informal discussion, we call two instances of a search problem *equivalent* if they have the same sets of solutions.

1.1.1. *Deterministic vs. Randomized Private Algorithms*

Consider first requiring any private algorithm $\mathcal{A}$ to be *deterministic*. As such, for each input, it consistently selects one of the solutions, and hence subsequent applications

[2] This requirement was first coined in the context of private approximation of functions [11] and later extended to search problems.

of the algorithm on the same (or equivalent) inputs do not reveal further information. A possible choice is to output the lexicographically first solution. This choice is computationally feasible for several polynomially solvable search problems such as the problem of finding a solution for a linear system, and stable marriage (using the stable marriage with restrictions algorithm [10]).[3]

Deterministic private search algorithms, however, leak definite information, which (depending on the application) may turn out to be crucial. Consider, for example, a database of computer science papers and their authors that on a query—a paper's title—returns one of the paper's authors' names. Whenever the deterministic private search algorithm answers differently on two instances—papers x and y—this serves as a definite indication that the author lists for x and y are not identical, even if the author lists are similar. If, furthermore, the deterministic private search algorithm returns the lexicographically first solution, it rules out all solutions that are ordered below it. For example, the answer to the query instance "How Should We Solve Search Problems Privately?" reveals that none of its authors' names starts with an "A".

Next, consider a *randomized* algorithm $\mathcal{A}$ which on input x selects an answer from the set of solutions according to a specific distribution (this distribution does not depend on the specific instance but may depend on the solution set). A natural choice here is to pick a solution uniformly at random. Randomized private algorithms may be advantageous to deterministic private algorithms, since the information they leak is potentially "blurred." In our database example, if papers x, y have similar author lists, then the resulting output distributions would be close. On the other hand, when applied repeatedly on the same instance, there is a potential for an increased leakage (this clearly is the case with our database example, since repeated querying yields a learning of all the authors of a paper in a coupon collecting manner). Another example, where the solution space is exponential and can be completely learned using a polynomial number of repeated evaluations of a randomized private search algorithm, is the problem of finding a solution for a linear system of equations; the number of revealed solutions grows roughly exponentially in the number of invocations, until the entire solution space is revealed.

We note that the benefits and disadvantages of deterministic and randomized algorithms are generally incomparable. Moreover, there exist problems for which an algorithm outputting uniformly selected solutions exists, but no deterministic private algorithm exists (under standard assumptions), and vice versa (see Sect. 5).

1.1.2. *Framework: Seeded Algorithms*

In the following, we restrict our attention to what we call *seeded algorithms*. The idea of seeded algorithms is not new—these are deterministic algorithms that get as input a "random" seed s and an instance x. If the seed s is selected at random the first time the seeded algorithm is invoked, subsequent invocations on the same input may be answered consistently. A seeded algorithm allows selecting a random solution for each instance (separately), while preventing abuse of repeated queries. Arguably, seeded algorithms

[3] The recent protocols of [12,16] also output a deterministic solution—the outcome of the Gale–Shapley algorithm [13]. However, the Gale–Shapley algorithm is not a private search algorithm.

are less desirable than algorithms that do not need to maintain any state information.[4] However, we note that the state information of seeded algorithms is rather easy to maintain, since the parties do not need to keep a log of previously answered queries, and hence their state does not grow with the number of queries. In that, the usage of seeded algorithms is similar to that of pseudorandom functions.

1.1.3. *Our Results*

To focus on the choice of a function for solving a search problem, we abstract out the implementation details of the underlying secure multiparty setting (similarly to [1,2, 11,17]). Our results directly apply to a client–server setup, where the server is willing to let the client learn a solution to a specific search problem applied to its input. They similarly apply to the setup of a distributed multiparty computation where the parties share an instance x using a secret sharing scheme, as it can be reduced to a client–server setup using secure function evaluation protocols [3,9,15,21]. In the general setup of distributed multiparty computation, however, one may also consider definitions that allow revealing information to a party implied by its individual input.

Equivalence-ProtectingAlgorithms Equivalence-protecting algorithms are seeded algorithms that choose a uniformly random answer for each class of equivalent instances. Given the seed, the output is deterministic and respects equivalence of instances—an access to an equivalence-protecting algorithm $\mathcal{A}_{\mathcal{P}}$ for a problem $\mathcal{P}$ simulates an access to a random oracle for $\mathcal{P}$ that answers consistently on inputs with the same solutions.[5]

To some extent, equivalence-protecting algorithms enjoy benefits of both the naïve privacy notions discussed above, deterministic and randomized private algorithms: (i) there is a potential for not giving "definite" information; and (ii) leakage is not accumulated with repeated queries. However, equivalence-protecting algorithms do allow distinguishing instances even when their solution sets are very close.

Recalling our database example, we note that equivalence protection may give better privacy guarantees than the examples of private search discussed in Sect. 1.1.1. The following toy "covert channel" problem is another example that demonstrates how equivalence protection can give better privacy guarantees than private search. An instance of the problem consists of a graph $G = (V, E)$, and a solution for an instance G consists of a matching M in G and a string C (whose content is not restricted). An equivalence-protecting algorithm should return a (pseudo) random string in C, rendering it useless for an attacker. On the other hand, a private search algorithm may put in C any function of the entire set of matchings for G, and hence the outcome string C may inadvertently turn into a channel leaking information about G. For instance, it may be that the private algorithm reveals one matching of G in M and another one in C. Of course, in real applications the covert channel may not be as visible as in this toy problem.

In Sect. 3 we reduce the problem of designing an equivalence-protecting algorithm for a search problem to that of (i) designing a deterministic algorithm for finding a

[4] In secure multiparty computation, the parties should jointly generate a random seed and then work with this shared seed in subsequent executions of the algorithm. In a client-server setup, the server should generate the seed the first time it is invoked and use it in future invocations.

[5] Such a random oracle can be thought of as an ideal model solution to the problem, which the equivalence-protecting algorithm is required to emulate.

canonical representative of the equivalence class; (ii) designing a randomized private algorithm returning a uniformly chosen solution; and (iii) the existence of pseudorandom functions. We then show how to use this to construct an equivalence-protecting algorithm for what we call "monotone search problems," a wide class of functions including perfect matching in bipartite graphs and shortest path in directed graphs. We further demonstrate the power of our general construction by showing an equivalence-protecting algorithm for solving a system of linear equations over a finite field.

Resemblance-Preserving Algorithms Our second strengthening of the requirements on a private search algorithm addresses the problem of distinguishing nonequivalent instances with similar solution sets. Similarly to equivalence-protecting algorithms, resemblance-preserving algorithms choose a random solution for each set of equivalence instances. However, here the choices of the output for nonequivalent instances are highly correlated so that pairs of instances that have close output sets are answered identically with high probability.

In Sect. 4 we present a generic construction of resemblance-preserving algorithms for any search problem whose output space admits a pairwise independent family of permutations where the minimum of a permuted solution set can be computed efficiently. Examples of such search problems include finding roots or non-roots of a polynomial, solving a system of linear equations over a finite field, finding a point in a union of rectangles in a fixed dimensional space, and finding a satisfying assignment for a DNF formula. It is interesting to note that for the last problem, finding an efficient equivalence-protecting algorithm implies $\mathrm{RP} = \mathrm{NP}$.

Summary We present two definitions that try to answer the particular concerns raised in Sect. 1.1.1 above and (at least intuitively) seem suitable for different applications. We provide technical tools to achieve these definitions and identify generic classes, as well as specific examples, of search problems where our tools can be used to yield private search algorithms with the desired properties.

It is interesting to note that the definition of equivalence-protecting algorithms is more "cryptographic," whereas the definition of resemblance-preserving algorithms is more "combinatorial." Equivalence-protecting is defined via using the ideal world vs. real world paradigm, and for many problems (including all our examples), the existence of an equivalence-protecting algorithm implies computational hardness—the existence of pseudorandom functions. The definition of resemblance-preserving algorithms does not use the ideal world vs. real world paradigm, and most of our examples only make use of combinatorial tools.

The main conceptual contribution of this work is in putting forward the need to study private computation of search problems (where a non-private solution is known). We also initiate a study of which privacy requirements are tractable for specific problems. Our main technical contributions are in the tools and algorithms presented in Sect. 4 for resemblance-preserving algorithms.

Other variants of private search have been researched in previous work. A variant where privacy is preserved w.r.t. instances that share the same set of *approximate* solutions was presented in [2,18], and a variant where the approximation algorithm is allowed to leak a well-quantified amount of information was presented in [1,17]. We note that unlike these two variants, which are relaxations of private search, the notions we examine in this work are strictly stronger than private search.

1.2. *Organization*

We start in Sect. 2 with definitions of search problems and the minimal privacy requirement of [1]. We define equivalence-protecting algorithms and resemblance-preserving algorithms and show the basic techniques for constructing them in Sect. 3 and in Sect. 4, respectively. We discuss the relationships between the classes of private algorithms in Sect. 5 and conclude in Sect. 6.

2. Definitions

We define a search problem as a function assigning to an instance $x \in \{0,1\}^n$ a solution set $\mathcal{P}_n(x)$. Two instances of a search problem are *equivalent* if they have exactly the same solution set. More formally:

Definition 2.1 (Search Problems). A search problem is an ensemble $\mathcal{P} = \{\mathcal{P}_n\}_{n\in\mathbb{N}}$ such that

$$\mathcal{P}_n : \{0,1\}^n \to 2^{\{0,1\}^{q(n)}}$$

for some positive polynomial $q(n)$.

Definition 2.2 (Instance Equivalence). For a search problem $\mathcal{P}$, the equivalence relation $\equiv_\mathcal{P}$ includes all pairs of instances $x, y \in \{0,1\}^n$ such that $\mathcal{P}_n(x) = \mathcal{P}_n(y)$.

We now recall the *minimal* definition of private search algorithms from [1]. All our privacy definitions will be stronger—an algorithm that satisfies one of Definitions 2.4, 2.5, 3.2, or 4.2 trivially satisfies Definition 2.3.

Definition 2.3 (Private Search Algorithms [1]). A probabilistic polynomial time algorithm $\mathcal{A}_\mathcal{P}$ is a private search algorithm for $\mathcal{P}$ if

1. $\mathcal{A}_\mathcal{P}(x) \in \mathcal{P}_n(x)$ for all $x \in \{0,1\}^n$, $n \in \mathbb{N}$ (that is, the algorithm always returns a valid solution); and
2. For every probabilistic polynomial-time algorithm $\mathcal{D}$ and for every positive polynomial $q(\cdot)$, there exists some $n_0 \in \mathbb{N}$ such that, for every $x, y \in \{0,1\}^*$ satisfying $x \equiv_\mathcal{P} y$ and $|x| = |y| \geq n_0$,

$$\left|\Pr[\mathcal{D}(\mathcal{A}_\mathcal{P}(x), x, y) = 1] - \Pr[\mathcal{D}(\mathcal{A}_\mathcal{P}(y), x, y) = 1]\right| \leq \frac{1}{q(|x|)}.$$

   That is, when $x \equiv_\mathcal{P} y$, no probabilistic polynomial time algorithm $\mathcal{D}$ can distinguish whether the input of $\mathcal{A}_\mathcal{P}$ is x or y.

Two natural special cases are of deterministic private algorithms and output-sampling algorithms:

Definition 2.4 (Deterministic Private Algorithms). Let $\mathcal{P} = \{\mathcal{P}_n\}_{n\in\mathbb{N}}$ be a search problem. A deterministic algorithm $\mathcal{A}$ is called a *deterministic private* algorithm for $\mathcal{P}$ if (i) $\mathcal{A}_{\mathcal{P}}(x) \in \mathcal{P}_n(x)$ for all $x \in \{0,1\}^n$, $n \in \mathbb{N}$; and (ii) $\mathcal{A}(x) = \mathcal{A}(y)$ for every $x, y \in \{0,1\}^*$ such that $x \equiv_{\mathcal{P}} y$.

Definition 2.5 (Output-Sampling Algorithms). Let $\mathcal{P} = \{\mathcal{P}_n\}_{n\in\mathbb{N}}$ be a search problem. A randomized algorithm $\mathcal{A}$ is called an *output-sampling* algorithm for $\mathcal{P}$ if for every $x \in \{0,1\}^n$, the distribution $\mathcal{A}(x,r)$ is computationally indistinguishable from $\text{Unif}_{\mathcal{P}(x)}$, the uniform distribution on the possible outputs on x. That is, for every family of polynomial-size circuits with oracle access $\{C_n\}_{n\in\mathbb{N}}$, every polynomial $p(\cdot)$, and all sufficiently large n's,

$$\left|\Pr\left[C_n^{\mathcal{A}(x,r)}\left(1^n\right)=1\right]-\Pr\left[C_n^{\text{Unif}_{\mathcal{P}(x)}}\left(1^n\right)=1\right]\right| < \frac{1}{p(n)}.$$

We proceed to the standard definition of (nonuniformly) pseudorandom functions from binary strings of size n to binary strings of size $\ell(n)$, where $\ell(\cdot)$ is some fixed polynomial.

Definition 2.6 (Pseudorandom Functions [14]). Let $\ell(\cdot)$ be some fixed polynomial and $F = \{F_n\}_{n\in\mathbb{N}}$ be a function ensemble, where F_n is a set of functions from $\{0,1\}^n$ to $\{0,1\}^{\ell(n)}$ for every n. The function ensemble F is called *pseudorandom* if for every family of polynomial-size circuits with oracle access $\{C_n\}_{n\in\mathbb{N}}$, every polynomial $p(\cdot)$, and all sufficiently large n's,

$$\left|\Pr\left[C_n^{F_n}\left(1^n\right)=1\right]-\Pr\left[C_n^{H_n}\left(1^n\right)=1\right]\right| < \frac{1}{p(n)},$$

where $H = \{H_n\}_{n\in\mathbb{N}}$ is the uniform function ensemble over the functions from $\{0,1\}^n$ to $\{0,1\}^{\ell(n)}$.

Finally, we define *seeded* algorithms, which are central to our constructions.

Definition 2.7 (Seeded Algorithms). A seeded algorithm $\mathcal{A}$ is a deterministic polynomial time algorithm taking two inputs x and s_n where $|x| = n$ and $|s_n| = p(n)$ for some polynomial $p()$. The distribution induced by a seeded algorithm on an input x is the distribution on outcomes $\mathcal{A}(x, s_n)$, where s_n is chosen uniformly at random from $\{0,1\}^{p(|x|)}$.

Informally, a seeded algorithm is private if it is a deterministic private algorithm with high probability over the choices of the seed s_n, i.e., $\mathcal{A}(x, s_n) = \mathcal{A}(y, s_n)$ for all but a negligible fraction of the seeds $s_n \in \{0,1\}^{p(|x|)}$ whenever $x \equiv_{\mathcal{P}} y$.

3. Equivalence-Protecting Algorithms

In this section we suggest a definition of private algorithm for a search problem and supply efficient algorithms satisfying this definition for a broad class of problems. The privacy guarantee we introduce enjoys some of the advantages of both deterministic and random algorithms. Based on the existence of pseudorandom functions, it provides solutions that look random but do not leak further information while executed repeatedly on inputs that are equivalent. In order to suggest appropriate privacy definitions for secure computation of a search problem, we need to picture how such a computation would take place in an ideal world. The following two definitions capture random sampling of an answer that depends only on the solution set (and not on the specific input).

Definition 3.1 (Private Oracles). Let $\mathcal{P} = \{\mathcal{P}_n\}_{n\in\mathbb{N}}$ be a search problem and q be the polynomial such that $\mathcal{P}_n : \{0,1\}^n \to 2^{\{0,1\}^{q(n)}}$. We say that for a given $n \in \mathbb{N}$, an oracle $O_n : \{0,1\}^n \to \{0,1\}^{q(n)}$ is private with respect to $\mathcal{P}_n$ if

1. For every $x \in \{0,1\}^n$, it holds that $O_n(x) \in \mathcal{P}_n(x)$. That is, O_n returns correct answers.
2. For every $x, x' \in \{0,1\}^n$, it holds that $x \equiv_{\mathcal{P}} x'$ implies $O_n(x) = O_n(x')$. That is, O_n satisfies the privacy requirement of Definition 2.3.

An oracle that is private with respect to $\mathcal{P}$ represents one possible functionality that solves the search problem and protects the equivalence relation. We define an algorithm to be equivalence-protecting if it cannot be efficiently distinguished from a random oracle that is private with respect to $\mathcal{P}$.

Definition 3.2 (Equivalence-Protecting Algorithms). Let $\mathcal{P} = \{\mathcal{P}_n\}_{n\in\mathbb{N}}$ be a search problem. An algorithm $\mathcal{A}(\cdot,\cdot)$ is equivalence protecting with respect to $\equiv_{\mathcal{P}}$ if for every family $\{\mathcal{D}_n\}_{n\in\mathbb{N}}$ of polynomial size distinguisher circuits with oracle access, for every polynomial p, and for all sufficiently large n's,

$$\left|\Pr\left[\mathcal{D}_n^{O_n}\left(1^n\right)=1\right]-\Pr\left[\mathcal{D}_n^{\mathcal{A}(\cdot,s_n)}\left(1^n\right)=1\right]\right| < \frac{1}{p(n)},$$

where the first probability is over the uniform distribution over oracles O_n that are private with respect to $\mathcal{P}$; the second probability is over the uniform distribution over the choices of the seed s_n for the algorithm $\mathcal{A}$.

In the above definition we arbitrarily choose the uniform distribution over private oracles. This choice is partly because using the uniform distribution is common in many sampling algorithms, see, e.g., [19]. However, for some applications, other distributions might be preferred; Definition 3.2 can be adjusted to such scenarios by requiring an appropriate distribution over the private oracles O_n.

3.1. *Canonical Representatives*

The following definition identifies a class of algorithms that will be helpful in constructing equivalence-protecting algorithms for various search problems. These algorithms return a representative element for every equivalence class of the search problem $\mathcal{P}$.

Definition 3.3 (Canonical Representative Algorithms). Let $\mathcal{P} = \{\mathcal{P}_n\}_{n\in\mathbb{N}}$ be a search problem. An algorithm $\mathcal{A}$ is a *canonical representative* algorithm for $\mathcal{P}$ if (i) $x \equiv_{\mathcal{P}} \mathcal{A}(x)$

```
Algorithm General Equivalence-Protecting

INPUT: An instance $x \in \{0,1\}^n$ and a seed $s_n$ for a family of pseudorandom functions $F = \{F_n\}_{n\in\mathbb{N}}$.
OUTPUT: A solution sol $\in \mathcal{P}_n(x)$.

  1. Compute $y = \mathcal{A}_{\text{rep}}(x)$.
  2. Compute $r = f_{s_n}(y)$.
  3. Output sol $= \mathcal{A}_{\text{sample}}(y, r)$.
```

Fig. 1. Algorithm `General Equivalence-Protecting`.

for every $x \in \{0,1\}^n$; and (ii) for every $x, y \in \{0,1\}^n$, it holds that $\mathcal{A}(x) = \mathcal{A}(y)$ iff $x \equiv_{\mathcal{P}} y$.

We reduce the problem of designing an equivalence-protecting algorithm for a search problem into designing a canonical representative algorithm and an output-sampling algorithm for the problem. The construction is based on the existence of pseudorandom functions. Let $F = \{F_n\}_{n\in\mathbb{N}}$ be an ensemble of pseudorandom functions from $\{0,1\}^n$ to $\{0,1\}^{\ell(n)}$, where $\ell(\cdot)$ is a polynomial that bounds the number of random bits used by the output-sampling algorithm. We denote by $f_{s_n}(x)$ the output of the function indexed by s_n on an input $x \in \{0,1\}^n$.

Theorem 3.4. *If a search problem $\mathcal{P}$ has an efficient output-sampling algorithm and an efficient canonical representative algorithm, then the seeded algorithm* `General Equivalence-Protecting` *presented in Fig. 1 is an efficient equivalence-protecting algorithm for $\mathcal{P}$.*

Proof. Let $\mathcal{A}(\cdot,\cdot)$ be the algorithm in Fig. 1. We show that given a family of distinguisher circuits $\{\mathcal{D}_n\}_{n\in\mathbb{N}}$ such that for infinitely many n,

$$\left|\Pr\left[\mathcal{D}_n^{O_n}(1^n) = 1\right] - \Pr\left[\mathcal{D}_n^{\mathcal{A}(\cdot,s)}(1^n) = 1\right]\right|$$

is noticeable (where O_n is a uniformly chosen private oracle with respect to $\mathcal{P}$), we can construct a family of circuits $\{C_n\}_{n\in\mathbb{N}}$ that distinguishes executions with oracle access to a pseudorandom function $f_{s_n}(\cdot)$ from executions with oracle access to a truly random function $H_n(\cdot)$.

The description of C_n is simple: C_n follows $\mathcal{D}_n$. Whenever $\mathcal{D}_n$ is making an oracle call on an input x, C_n simulates it as follows: (i) it computes $y = \mathcal{A}_{\text{rep}}(x)$; (ii) it computes the answer z of the oracle call on y (i.e., z is either an answer of a random function, namely, $H_n(y)$, or an answer of a pseudorandom function, namely, $f_{s_n}(y)$); (iii) it computes $\mathcal{A}_{\text{sample}}(y, z)$.

Note that $C_n^{F_n}$ behaves identically to $\mathcal{D}_n^{\mathcal{A}(\cdot,s)}$. We next show that the difference

$$\left|\Pr\left[\mathcal{D}_n^{O_n}(1^n) = 1\right] - \Pr\left[C_n^{H_n}(1^n) = 1\right]\right|$$

is negligible. Note that every call of $\mathcal{D}_n$ to $O_n(x)$ results in a random element in $\mathcal{P}(x)$. Since C_n applies $\mathcal{A}_{\text{sample}}(y, H_n(y))$, where $y \equiv_{\mathcal{P}} x$ and H_n is a randomly chosen function, by the properties of the output-sampling algorithm $\mathcal{A}_{\text{sample}}$, the output is indistinguishable from a random element in $\mathcal{P}(x)$. Hence, the distance

$$\left|\Pr\left[C_n^{F_{s_n}}(1^n) = 1\right] - \Pr\left[C_n^{H_n}(1^n) = 1\right]\right|$$

must be noticeable for infinitely many n, in a contradiction to the fact that F is an ensemble of pseudorandom functions. □

3.2. *Equivalence-Protecting Algorithms for Monotone Search Problems*

In view of Theorem 3.4, the construction of an equivalence-protecting algorithm for a given search problem is reduced to finding a canonical representative algorithm and an output-sampling algorithm for the problem. We focus on search problems in which an output is a subset of the input satisfying some property (e.g., the input is a graph, and the output is any subgraph satisfying some property). We reduce the design of a canonical representative algorithm into deciding whether an input element is contained in some possible output.

Definition 3.5 (Monotone Search Problems). Let $\mathcal{P}$ be a search problem and view the inputs to $\mathcal{P}_n$ as subsets of $[n]$. We say that $\mathcal{P}$ is a *monotone search problem* if there exists a set $S \subseteq 2^{[n]}$ such that $\mathcal{P}_n(X) = 2^X \cap S$ for every input $X \subseteq [n]$. That is, there is a global set S of solutions, and the solutions for X are the solutions in S that are contained in X.

For example, the problem of finding a perfect matching in a bipartite graph is monotone. The global set of solutions S consists of all the graphs whose edges form exactly a perfect matching. For every bipartite graph G, the set of solutions on G is the set of graphs in S (i.e., perfect matching graphs) that are subgraphs of G.

Definition 3.6 (Relevant Elements). Let $\mathcal{P}$ be a monotone search problem and $X \subseteq [n]$ be an input to $\mathcal{P}_n$. We say that $i \in X$ is *relevant* to X if there is an output $Y \in \mathcal{P}_n(X)$ such that $i \in Y$. We denote by $R(X)$ the set of elements relevant to X.

In the perfect matching example, an edge is relevant if it appears in some perfect matching of the graph. The following claim shows that computing $R(X)$ efficiently from X is sufficient to get a canonical representative algorithm.

Claim 3.7. *Let $\mathcal{P}$ be a monotone search problem and $X, Y \subseteq [n]$ be inputs of P_n. Then* (i) $X \equiv_P R(X)$; *and* (ii) $X \equiv_P Y$ *if and only if* $R(X) = R(Y)$.

Proof. (i) We show that X and $R(X)$ have the same sets of solutions. Let Y be a solution to X. Every $i \in Y$ is relevant to X and thus $i \in R(X)$. Hence $Y \subseteq R(X)$, and therefore Y is a solution to $R(X)$. For the other direction, let Y be a solution to $R(X)$. Obviously $R(X) \subseteq X$, and thus $Y \subseteq X$; therefore Y is a solution to X. (ii) Assume $X \equiv_P Y$ and let $i \in R(X)$. Then $i \in Z$, where Z is a solution to X. Since $X \equiv_P Y$, we get that Z is also a solution to Y, and thus $i \in R(Y)$. The other direction is immediate from (i) and the transitivity of $\equiv_P$. □

3.3. *Applications of the Construction*

We next introduce equivalence-protecting algorithms for some well-known search problems.

Example 3.8 (Perfect Matching in Bipartite Graphs). Consider the problem of finding a perfect matching in a bipartite graph $G = \langle G, E \rangle$. To decide whether an input edge $\langle u, v \rangle$ is relevant we do the following: (i) Denote by G' the graph that results from deleting u, v and all the edges adjacent to them from G. (ii) Check whether there is a perfect matching in G'. Evidently, $\langle u, v \rangle$ is relevant to G if and only if G' has a perfect matching. Hence, perfect matching has an efficient canonical representative algorithm.

As an output-sampling algorithm, we use the algorithm of Jerrum et al. [19]. The algorithm samples a perfect matching of a bipartite graph from a distribution that is statistically close to uniform. Therefore, we have both a canonical representative and output-sampling algorithm for perfect matching, and thus by Theorem 3.4, we get that perfect matching has an efficient equivalence-protecting algorithm.

It is natural to ask whether the reliance on pseudorandom functions is inherent for this problem. This is indeed true, since the existence of a equivalence-protecting algorithm implies the existence of pseudorandom functions. We next show how to construct an ensemble of pseudorandom functions $F = \{F_n\}$, where $F_n = \{f_s : \{0,1\}^n \to \{0,1\}^{m(n)}\}$, from an equivalence-protecting algorithm $\mathcal{A}(\cdot, s)$. To apply $f_s()$ on $x \in \{0,1\}^n$, construct a bipartite graph G consisting of two disjoint subgraphs: (i) a bipartite subgraph containing a perfect matching uniquely encoding x (i.e., no two of the constructed graphs share an equivalence class); and (ii) a bipartite subgraph consisting of $m(n)$ copies of the complete bipartite graph $K_{2,2}$ (note that in a perfect matching, there are exactly two matching possibilities for each copy of $K_{2,2}$). Apply $\mathcal{A}(\cdot, s)$ to G and output the $m(n)$ bits corresponding to the chosen matchings of the copies of $K_{2,2}$.

Example 3.9 (Linear Algebra). Let n and m be positive integers, $\mathbb{F}$ be a finite field, M be an $n \times m$ matrix over $\mathbb{F}$, and $v \in \mathbb{F}^n$. Consider the problem of solving the system $My = v$. Since this problem is not monotone, we need to design both the canonical representative algorithm and the output-sampling algorithm. As a canonical representative algorithm, simply perform the Gaussian elimination procedure on the system, bringing it to its reduced row echelon form. An elementary linear algebra argument shows that if two systems have the same sets of solutions, then they have the same reduced row echelon form. We now show a simple output-sampling algorithm for the problem: Compute an arbitrary solution $y_0 \in \mathbb{F}^m$ satisfying $My_0 = v$. Compute $k = \text{rank}(M)$ and compute an $m \times (n-k)$ matrix K representing the kernel of the matrix M. Randomly pick a vector $r \in \mathbb{F}^{n-k}$ and output $w = y_0 + Kr$. Again, elementary linear algebra argument shows that w is a random solution to the system $My = v$.

Example 3.10 (Shortest Path). Consider the problem of finding a shortest path from a vertex s to a vertex t in a directed graph G. In this case there is no global set of solutions, since a path can be an appropriate solution for one graph, while in another graph there may be shorter paths. However, there exists a canonical representative algorithm for the problem; this algorithm on input G outputs a layered graph G' consisting of the set of edges that appear in any shortest path from s to t in G (checking whether an edge is relevant for G can be done, e.g., using breadth-first search). To sample a random solution, do: (i) Starting from t, compute for every $v \in V$ of distance i from t the number of shortest paths from v to t, by summing the number of shortest paths from v's neighbors of distance $i-1$ to t. (ii) Starting from s, pick the vertices on the path randomly, where the probabilities are weighted according to the number of paths computed in (i). Hence, by Theorem 3.4, a shortest path has an efficient equivalence-protecting algorithm.

Similar ideas are applicable for finding a shortest path in a *weighted* directed graph with positive weights. Here, however, we do not apply Theorem 3.4 directly. The equivalence-protecting algorithm, given a directed graph $G = \langle G, E\rangle$, a weight function $w : E \to \mathbb{R}^+$, two vertices s, t, and a seed s_n for a family of pseudorandom functions, does the following: (i) Construct a directed graph $G' = \langle V, E'\rangle$, where E' is the set of edges that appear in at least one shortest path from s to t. Note that G' is a non-weighted acyclic directed graph; (ii) Compute $r = f_{s_n}(G')$; (iii) Sample, using r, a random path from s to t in the acyclic graph G' with uniform distribution and output this path. Note that in step (iii) we sample a random path and not a random shortest path. This example is different in the fact that the canonical input we use in steps (ii) and (iii) is an instance of a problem that is slightly different than the original problem.

4. Resemblance-Preserving Algorithms

We now strengthen the requirement on private algorithm in an alternative manner to the definition of equivalence-protecting algorithms presented in Sect. 3. The motivation for the definition in this section is that the output of the algorithm should not distinguish between inputs with similar sets of solutions. While this requirement is met by a randomized algorithm that outputs a uniform solution, it cannot in general be satisfied by a deterministic algorithm for search problems—for all inputs $x, y \in \{0,1\}^n$, the algorithm would have to output the same "solution" on x and y if there exists inputs $x = z_0, z_1, \ldots, z_t = y$, all in $\{0,1\}^n$, such that $\mathcal{P}_n(z_i) \cap \mathcal{P}_n(z_{i+1}) \neq \emptyset$; this would, in many cases, contradict the correctness of the search algorithm. As we want an algorithm that does not leak more information on repeated executions, we put forward a definition of resemblance-preserving algorithms, which are seeded algorithms that protect inputs with similar sets of outputs.

To measure the similarity between the sets of outputs, we use resemblance between sets (a.k.a. Jaccard index), a notion used in [6–8] and seeming to capture the informal notion of "roughly the same." For example, in [6,7] resemblance between documents was successfully used for clustering documents.

Definition 4.1 (Resemblance). Let U be a set, and let $A, B \subseteq U$. Then the resemblance between A and B is defined to be

$$r(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

Note that $0 \le r(A, B) \le 1$, where $r(A, B) = 0$ iff A and B are disjoint and $r(A, B) = 1$ iff $A = B$. For a search problem $\mathcal{P}$, we will consider the resemblance between solution sets $\mathcal{P}_n(x), \mathcal{P}_n(y)$ of $x, y \in \{0, 1\}^n$. Informally, a (perfect) resemblance-preserving algorithm is a seeded algorithm that returns the same output for x and y with probability of at least the resemblance between $\mathcal{P}_n(x)$ and $\mathcal{P}_n(y)$.

Definition 4.2 (Resemblance-Preserving Algorithms). An algorithm $\mathcal{A}(\cdot, \cdot)$ is resemblance-preserving with respect to $\mathcal{P}$ if:

1. For every probabilistic polynomial-time algorithm $\mathcal{D}$ and every polynomial $p(\cdot)$, there exists some $n_0 \in \mathbb{N}$ such that for every $x \in \{0, 1\}^*$ satisfying $|x| > n_0$,

$$\left|\Pr\left[\mathcal{D}\left(x, \mathcal{A}(s_n, x)\right) = 1\right] - \Pr\left[\mathcal{D}\left(x, \mathrm{Unif}\left(\mathcal{P}_n(x)\right)\right) = 1\right]\right| \le \frac{1}{p(|x|)}.$$

   The probability is taken over the random choice of the seed s_n and the randomness of $\mathcal{D}$. Informally, taking the probability over the seed, the output of $\mathcal{A}_{\mathcal{P}}$ on x is indistinguishable from the uniform distribution on $\mathcal{P}_n(x)$.

2. There exists a constant $c > 0$ such that for all $x, y \in \{0, 1\}^*$ satisfying $|x| = |y|$,

$$\Pr\left[\mathcal{A}(s_n, x) = \mathcal{A}(s_n, y)\right] \ge c \cdot r\left(\mathcal{P}_n(x), \mathcal{P}_n(y)\right).$$

   The probability is taken over the random choice of s_n. That is, the probability that $\mathcal{A}$ returns the same output on two inputs is at least some constant times the resemblance between $\mathcal{P}_n(x)$ and $\mathcal{P}_n(y)$.

3. If $x \equiv_{\mathcal{P}} y$, then $\mathcal{A}(s_n, x) = \mathcal{A}(s_n, y)$ for all seeds s_n. That is, if x and y are equivalent, then $\mathcal{A}$ always returns the same output on x and on y.

If $c = 1$ in the above Requirement 2, then $\mathcal{A}(\cdot, \cdot)$ is *perfect* resemblance-preserving with respect to $\mathcal{P}$.

We note that our choice in Requirement 2 above that the probability should be at least linear in the resemblance is somewhat arbitrary. A definition of a similar flavor results by requiring the probability to be lower bounded by some monotone function of the resemblance. Requirement 3 can be relaxed to all but a negligible fraction of the seeds.

Resemblance-Preserving vs. Equivalence-Protecting We note that the requirements in Definitions 3.2 and 4.2 are incomparable (we show in Sect. 5.3 that there exist search problems that admit resemblance-preserving but no equivalence-protecting algorithms, and vise versa). In particular, we note that unlike Definition 3.2, in the definition of resemblance-preserving algorithms we do not know how to formulate privacy using the real world vs. ideal world paradigm. One of the consequences of this difference is that it is possible to design resemblance-preserving algorithms without needing cryptographic assumptions. In our constructions, for example, we only use pairwise independent permutations. Furthermore, Definition 4.2 does not prevent partial disclosure or even full disclosure of the seed by the algorithm. This should be considered when using a resemblance-preserving algorithm.

Example 4.3 (Non-Roots of a Polynomial). We give an example demonstrating that perfect resemblance-preserving algorithms exist. Consider the following problem. The inputs are univariate polynomials of degree $d(n)$ over $\mathbb{F}_{2^n}$, where $d : \mathbb{N} \to \mathbb{N}$ is some fixed increasing function (e.g., $d(n) = n$). The set of solutions of a polynomial Q is the set of all points y which *are not* roots of Q, that is, $\{y \in \mathbb{F}_{2^n} : Q(y) \neq 0\}$. This problem arises, e.g., when we want to find points on which two polynomials disagree.

The seed s_n in the algorithm we construct is a random string of length $(d(n) + 1) \cdot n$ considered as a list of $d(n) + 1$ elements in $\mathbb{F}_{2^n}$. Since Q has at most $d(n)$ roots, there is an element in the list s_n that is not a root of Q. The algorithm on input Q returns the first element in s_n that is not a root of Q. We claim that this algorithm is resemblance-preserving. First, since the seed is chosen at random, the first element in the list that is not a root is a random non-root of Q. Second, consider two polynomials Q_1 and Q_2 with sets of non-roots Y_1 and Y_2, respectively. The algorithm returns the same non-root on both Q_1 and Q_2 if the first element in the list s_n from Y_1 is also the first element in the list s_n from Y_2. In other words, the algorithm returns the same non-root if the first element in the list s_n which is from the set $Y_1 \cup Y_2$ is from $Y_1 \cap Y_2$. The probability of this event is exactly $r(Y_1, Y_2) = |Y_1 \cap Y_2|/|Y_1 \cup Y_2|$.

4.1. *Generic Constructions of Resemblance-Preserving Algorithms*

We present our main tool for constructing resemblance-preserving algorithms—min-wise independent permutations. We will first show a general construction, which (depending on the search problem) may exhibit exponential time complexity. Then, we will present the main contribution of this section—a polynomial-time resemblance-preserving algorithm that is applicable for problems for which there is a pairwise independent family of permutations where we can efficiently compute the minimum on any set of solutions.

Definition 4.4 (Family of Min-wise Independent Permutations [8]). Let U be a set and $\mathcal{F} = \{\pi_s\}_{s \in S}$ be a collection of permutations $\pi_s : U \to U$. The collection $\mathcal{F}$ is a collection of min-wise independent permutations if $\Pr[\min(\pi_s(A)) = \pi_s(a)] = 1/|A|$ for all $A \subseteq U$ and all $a \in A$. The probability is taken over the choice of the seed s at uniform from S.

We will use the following observation that relates min-wise permutations and resemblance:

Observation 4.5 [8]. Let $\mathcal{F}$ be a family of min-wise independent permutations $\{\pi_s\}_{s \in S}$, where $\pi_s : U \to U$. Then $\Pr[\min(\pi_s(A)) = \min(\pi_s(B))] = r(A, B)$ for every sets $A, B \subseteq U$. The probability is taken over the choice of the seed s at uniform from S.

In Fig. 2, we describe Algorithm $\texttt{Minwise}_{\mathcal{P}}$ for a search problem $\mathcal{P}$, where $\mathcal{P}_n : \{0,1\}^n \to \{0,1\}^{q(n)}$. Using Observation 4.5, it is easy to see that Algorithm $\texttt{Minwise}_{\mathcal{P}}$ is perfectly resemblance-preserving.

However, Algorithm $\texttt{Minwise}_{\mathcal{P}}$ may be inefficient in several aspects:

```
Algorithm Minwise_P

INPUT: An instance $x \in \{0,1\}^*$ and a seed $s$ for a family of min-wise independent permutations
$\{\pi_s\}_{s \in S}$, where $\pi_s : \{0,1\}^{q(|x|)} \to \{0,1\}^{q(|x|)}$.
OUTPUT: A solution sol $\in \mathcal{P}_n(x)$.

  1. Let $A = \mathcal{P}_n(x)$.
  2. Output sol $\in A$ such that $\pi_s(\text{sol}) = \min \pi_s(A)$.
```

Fig. 2. Algorithm $\texttt{Minwise}_{\mathcal{P}}$.

1. Algorithm $\texttt{Minwise}_{\mathcal{P}}$ uses a family of min-wise independent permutations. It was shown in [8] that such families are of size $2^{\Omega(|U|)} = 2^{\Omega(2^{q(n)})}$ (where n is the input length), and hence the seed length $|s| = \Omega(2^{q(n)})$. However, for most purposes, the seed length may be reduced to polynomial by using pseudorandom permutations.[6]
2. Algorithm $\texttt{Minwise}_{\mathcal{P}}$ needs to compute the minimum element, according to π_s, in the solution set $\mathcal{P}_n(x)$. This is feasible when it is possible to enumerate in polynomial time the elements of $\mathcal{P}_n(x)$. However, to make $\texttt{Minwise}_{\mathcal{P}}$ feasible in cases where, for example, $\mathcal{P}_n(x)$ is of super-polynomial size, one needs to carefully use the structure of π_s and the structure of the underlying solution set space.

Example 4.6 (Roots of a Polynomial). As an example that Algorithm $\texttt{Minwise}_{\mathcal{P}}$ can be implemented efficiently, we consider the problem of finding roots of a polynomial. As in Example 4.3, the inputs are univariate polynomials of degree $d(n)$ over $\mathbb{F}_{2^n}$, where $d : \mathbb{N} \to \mathbb{N}$ is some fixed increasing function (e.g., $d(n) = n$). The set of solutions of a polynomial Q is the set of all points y which *are* roots of Q, that is, $\{y \in \mathbb{F}_{2^n} : Q(y) = 0\}$. Berlekamp [4] presented an efficient algorithm that finds roots of a polynomial over $\mathbb{F}_{2^n}$. We implement Algorithm $\texttt{Minwise}_{\mathcal{P}}$, where we use a family of pseudorandom permutations from $\mathbb{F}_{2^n}$ to $\mathbb{F}_{2^n}$ instead of the family of min-wise independent permutations.[7] Furthermore, since the number of roots of a polynomial of degree $d(n)$ is at most $d(n)$, we can use Berlekamp's algorithm to explicitly find all roots of the polynomial, apply the pseudorandom permutation to each root, and find for which root $\pi_s(y)$ obtains a minimum. The above algorithm can be generalized to any search problems whose entire sets of solutions can be generated efficiently.

Observation 4.7. If for a search problem $\mathcal{P}$, there is an algorithm that generates the set of solutions of an input of $\mathcal{P}$ whose running time is polynomial in the length of the input (and, in particular, the number of solutions in polynomial), then Algorithm $\texttt{Minwise}_{\mathcal{P}}$ can be efficiently implemented for $\mathcal{P}$.

[6] We need the family of pseudorandom permutations to be secure against a nonuniform adversary. Thus, for every long enough inputs x and y, a pseudorandom permutation must be min-wise. We omit further details as instead of using pseudorandom permutations we replace, in Sect. 4.2, min-wise independent permutations with pairwise independent permutations.

[7] If we do not want to use cryptographic assumptions, we can construct a nonperfect resemblance-preserving algorithm as in Sect. 4.2.

4.2. *Resemblance-Preserving Using Pairwise Independence*

To get around the above mentioned problems of implementing $\mathtt{Minwise}_{\mathcal{P}}$ for search problems with a super-polynomial number of solutions, we construct a nonperfect resemblance-preserving algorithm using pairwise independence permutations instead of min-wise independence.

Definition 4.8 (Family of Pairwise Independent Permutations). Let U be a set and $\mathcal{F} = \{\pi_s\}_{s\in S}$ be a collection of permutations $\pi_s : U \to U$. The collection $\mathcal{F}$ is a family of pairwise independent permutations if

$$\Pr\bigl[\pi_s(a) = c \wedge \pi_s(b) = d\bigr] = \frac{1}{|U|(|U|-1)}$$

for all $a, b \in U$ and $c, d \in U$, where $a \neq b$ and $c \neq d$. The probability is taken over the uniform choice of the seed s from S.

Theorem 4.9 [8]. *Let $\mathcal{F} = \{\pi_s\}_{s\in S}$ be a family of pairwise independent permutations, where $\pi_s : U \to U$. Then for every set $A \subseteq U$ and every $a \in A$,*

$$\frac{1}{2(|A|-1)} \leq \Pr\bigl[\min\bigl(\pi_s(A)\bigr) = \pi_s(a)\bigr] \leq \frac{2}{\sqrt{|A|-1}-1}.$$

The probability is taken over the uniform choice of the seed s from S.

For completeness, we give the proof of Theorem 4.9 in Appendix A. We get the following lemma:

Lemma 4.10. *Let $\mathcal{F} = \{\pi_s\}_{s\in S}$ be a family of pairwise independent permutations, where $\pi_s : U \to U$. Then for all sets $A, B \subseteq U$,*

$$\Pr\bigl[\min\bigl(\pi_s(A)\bigr) = \min\bigl(\pi_s(B)\bigr)\bigr] \geq \max\left(\frac{r(A,B)}{2}, 1 - \frac{2\cdot|A\Delta B|}{\sqrt{|A\cup B|-1}-1}\right).$$

The probability is taken over the uniform choice of the seed s from S.

Proof. To see the first bound, note that

$$\begin{aligned}
\Pr\bigl[\min\bigl(\pi_s(A)\bigr) = \min\bigl(\pi_s(B)\bigr)\bigr] &= \Pr\bigl[\min\bigl(\pi_s(A\cup B)\bigr) = \min\bigl(\pi_s(A\cap B)\bigr)\bigr] \\
&= \sum_{a\in A\cap B} \Pr\bigl[\min\bigl(\pi_s(A\cup B)\bigr) = \pi(a)\bigr] \\
&\geq \frac{|A\cap B|}{2(|A\cup B|-1)} \geq \frac{r(A,B)}{2}.
\end{aligned}$$

The second bound follows from the right inequality of Theorem 4.9:

$$\begin{aligned}
\Pr\bigl[\min\bigl(\pi_s(A)\bigr) = \min\bigl(\pi_s(B)\bigr)\bigr] &= 1 - \Pr\bigl[\min\bigl(\pi_s(A\cup B)\bigr) = \min\bigl(\pi_s(A\Delta B)\bigr)\bigr] \\
&\geq 1 - |A\Delta B|\frac{2}{\sqrt{|A\cup B|-1}-1}.
\end{aligned}$$

```
Algorithm Pairwise$_{\mathcal{P}}$

INPUT: An instance $x \in \{0,1\}^*$ and a seed $s$ for a family of pairwise independent permutations
$\{\pi_s\}_{s \in S}$, where $\pi_s : \{0,1\}^{q(|x|)} \to \{0,1\}^{q(|x|)}$.
OUTPUT: A solution sol $\in \mathcal{P}_n(x)$.

  1. Let $A = \mathcal{P}_n(x)$.
  2. Output sol $\in A$ such that $\pi_s(\text{sol}) = \min \pi_s(A)$.
```

Fig. 3. Algorithm Pairwise$_{\mathcal{P}}$.

□

In Fig. 3, we construct an algorithm Pairwise$_{\mathcal{P}}$ that is almost identical to Minwise$_{\mathcal{P}}$ of Fig. 2, where the family of min-wise permutations is replaced with a family of pairwise independent permutations. The following corollary follows directly from Lemma 4.10:

Corollary 4.11. *Algorithm* Pairwise$_{\mathcal{P}}$ *of Fig.* 3 *is resemblance-preserving.*

4.3. *Applications of the Pairwise Independence Construction*

We next show how to apply Algorithm Pairwise$_{\mathcal{P}}$ to a few search problems. Given a search problem, we need to choose the family of pairwise independent permutations such that the solution minimizing $\pi_s(A)$ can be computed efficiently. In our examples we use the following well-known family of pairwise independent permutations from $\mathbb{F}_q^n$ to $\mathbb{F}_q^n$ for some prime-power q:

$$\mathcal{L}_{q,n} \overset{\text{def}}{=} \left\{ Hy + b : H \text{ is an invertible } n \times n \text{ matrix over } \mathbb{F}_q \text{ and } b \in \mathbb{F}_q^n \right\}.$$

Claim 4.12. *The family $\mathcal{L}_{q,n}$ is a family of pairwise independent permutations.*

4.3.1. *Linear Algebra*

We show how to construct a resemblance-preserving algorithm for finding a solution of a system of equations (as considered in Example 3.9 for Equivalence-Protecting Algorithms).

Linear Algebra over $\mathbb{F}_2$ We assume that the system is over $\mathbb{F}_2$. In the next paragraph we generalize the result to every finite field. That is, the input is an $m \times n$ matrix M over $\mathbb{F}_2$ and a vector $v \in \mathbb{F}_2^m$, and a solution is a vector $y \in \mathbb{F}_2^n$ such that

$$My = v. \tag{1}$$

We apply Algorithm Pairwise$_{\mathcal{P}}$ for this problem using the family $\mathcal{L}_{2,n}$. That is, we choose a permutation at random, specified by H and b, and we need to find the lexicographically first z satisfying

$$z = Hy + b \tag{2}$$

for y satisfying $My = v$. We view $My = v$ and $z = Hy + b$ as a single system of linear equations with $2n$ unknowns, namely, $y = \langle y_1, \dots, y_n \rangle$ and $z = \langle z_1, \dots, z_n \rangle$. To find the value of z_1 in the lexicographically first z, we add the equation $z_1 = 0$ to the system of equations. If the new system has a solution, we keep the equation $z_1 = 0$ in the system and continue to find the value of z_2. Otherwise, we understand that $z_1 = 1$ in every solution of the original system of equations, and, in particular, in the lexicographically first z. In this case, we remove the equation $z_1 = 0$ from the system of equations and continue to find the value of z_2. To conclude, we find the lexicographically first z iteratively, where in iteration i we have already found the values of $z_1, \dots, z_{i-1}$, and we compute the value of z_i in the lexicographically first z as we found z_1. We continue these iterations until we find the lexicographically first z. Recall that $Hy + b$ is a permutation. Thus, once we found z, the solution y is uniquely defined and is easy to compute from the system of equations, that is, compute $y = H^{-1}(z - b)$.

Linear Algebra over Finite Fields We show that the construction described for linear algebra over $\mathbb{F}_2$ works for an arbitrary finite field $\mathbb{F}_q$ when we use $\mathcal{L}_{q,n}$ as the pairwise independent permutations. In the case of $\mathbb{F}_2$, when we ruled out the case $z_i = 0$, we immediately deduced that $z_i = 1$. In a first glance, it seems that generalizing the above construction to $\mathbb{F}_q$ would require checking if $x_i = a$ for every value of $a \in \mathbb{F}_q$. However, the next claim shows that this is not required.

Claim 4.13. *Let $Dw = c$ be a system of linear equations. Assume that there are two solutions v and w to the system such that $v_i \neq w_i$ for some i. Then the system has a solution u such that $u_i = 0$.*

Proof. Let

$$u \overset{\text{def}}{=} v - \frac{v_i}{v_i - w_i}(v - w).$$

First, since $v_i \neq w_i$, the vector u is properly defined. Second, $u_i = v_i - \frac{v_i}{v_i - w_i}(v_i - w_i) = 0$, as required. Third, u is a solution to the system of equations

$$\begin{aligned} Du &= D\left(v - \frac{v_i}{v_i - w_i}(v - w)\right) = Dv - \frac{v_i}{v_i - w_i}(Dv - Dw) \\ &= c - \frac{v_i}{v_i - w_i}(c - c) = c. \end{aligned}$$ □

Recall that the algorithm for finding the lexicographically first z iteratively adds the equation $z_i = 0$ if, after adding the equation, the system has a solution and does not add anything otherwise. By Claim 4.13, when the algorithm does not add the equation $z_i = 0$, there is a unique value for z_i in all the solutions. Thus, after the n iterations, the system has a unique solution y, z, and the algorithm computes y and returns it.

Union of Linear Spaces We want to use the resemblance-preserving algorithm for finding a solution of a system of linear equations to construct resemblance-preserving algorithms for other problems. That is, we want to represent the set of solutions of an instance of some search problem as a set of solutions of a system of linear equations. In our applications, we manage to represent the set of solutions of an instance as a *union* of polynomially many linear spaces represented by systems of linear equations over the same field. We next show how to construct a resemblance-preserving algorithm for such a union. That is, the input is a sequence $M_1, v_1, \ldots, M_\ell, v_\ell$, and a solution is a vector y such that $M_i y = v_i$ for at least one i.

```
Algorithm LinearAlgebraUnion

INPUT: A sequence $M_1, v_1, \ldots, M_\ell, v_\ell$ and a seed $H, b$.
OUTPUT: A vector $y$ such that $M_i y = v_i$ for at least one $i$.

  1. Find for each system of equations a solution $y_i$ such that $Hy_i + b$ is minimized amongst
     all vectors such that $M_i y = v_i$.
  2. Output $y_j$ such that $Hy_j + b = \min\{Hy_i + b : 1 \le i \le \ell\}$.
```

Theorem 4.14. *There is a resemblance-preserving algorithm for finding a solution in a union of polynomially many solution sets of systems of linear equations over the same field.*

4.3.2. *Points in a Union of Discrete Rectangles*

We show how to use the resemblance-preserving algorithm for linear algebra to construct resemblance-preserving algorithms for finding a point in a union of discrete rectangles. We construct such algorithms for two cases: (1) unions of rectangles in $[2]^n$, that is, DNF formulae, and (2) unions of rectangles in $[N]^d$ when d is fixed (however, N is not fixed).

Satisfying Assignment for a DNF Formula We show how to construct a resemblance-preserving algorithm for finding a satisfying assignment of a DNF formula. This follows from Theorem 4.14 and the following observations. First, the set of satisfying assignments of a single term is the set of solutions of a system of linear equations over $\mathbb{F}_2$:

- For every variable x_i that appears in the term without negation, add the equation $y_i = 1$.
- For every variable x_i that appears in the term with negation, add the equation $y_i = 0$.

Now, given a DNF formula with ℓ terms, a satisfying assignment to the formula is an assignment satisfying at least one of the terms in the formula, that is, it belongs to the union of solutions of the ℓ systems of linear equations constructed for each of the terms of the formula. Thus, by Theorem 4.14, we get a resemblance-preserving algorithm for finding a satisfying assignment of a DNF formula.

Points in a Union of Discrete Rectangles in a d-dimensional Space We show how to construct a resemblance-preserving algorithm for finding a point in a discrete rectangle. That is, for some fixed $d \in \mathbb{N}$ and for an integer $N \in \mathbb{N}$, our inputs are $2d$ elements $a^1, \ldots, a^d, b^1, \ldots, b^d \in [N]$ which represent a rectangle as follows:

$$R_{a^1,\ldots,a^d,b^1,\ldots,b^d} \stackrel{\text{def}}{=} \left\{\left\langle y^1, \ldots, y^d\right\rangle : \forall_{i\in[d]} a^i \le y^i \le b^i\right\}.$$

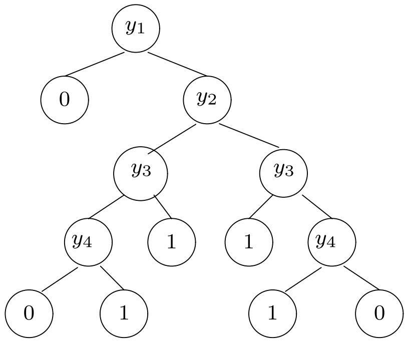

Fig. 4. A decision tree for $a \le y \le b$, where $a = (1, 0, 0, 1)$ and $b = (1, 1, 1, 0)$.

Let $n \overset{\text{def}}{=} \lceil \log N \rceil$; we represent a number $a \in [N]$ by an n-bit string $a_1, \ldots, a_n$, where $a = \sum_{i=1}^{n} a_i 2^{n-i}$. Note that, in this section, a^i is a string in $\{0, 1\}^n$, and a_i is the ith bit of a string a.

We solve the problem of finding a point in a rectangle by describing a decision tree[8] that accepts exactly the points in the rectangle. Since each decision tree with ℓ nodes can be represented by a DNF formula with at most ℓ terms, we can use the resemblance-preserving algorithm for DNF (described in the previous paragraph) to output a satisfying assignment to the formula, i.e., a point in the rectangle.

Let us start with the simple case where $d = 1$. That is, an input is two strings a and b, and a solution is a string $a \le y \le b$. Let $a < b$ and i_0 be the minimal index such that $a_i = 0$ and $b_i = 1$. A point y satisfies $a \le y \le b$ if $y_j = a_j = b_j$ for every $1 \le j \le i_0 - 1$ and one of the following conditions hold:

1. $y_{i_0} = a_{i_0}$ and $a \le y$, or
2. $y_{i_0} = b_{i_0}$ and $y \le b$.

The decision tree for this case is very simple; it is basically composed of two paths of internal nodes, and it has at most $2n$ leaves. We will not describe the tree but rather give an example in Fig. 4, where $a = (1, 0, 0, 1)$ and $b = (1, 1, 1, 0)$. In this example, if $y_1 = 0$, then $y < a$, thus, the root of the decision tree is labeled by the variable y_1, its left son is a leaf labeled by 0, and its right subtree is a decision tree checking if $(0, 0, 1) \le (y_2, y_3, y_4) \le (1, 1, 0)$. Furthermore, $a_2 \ne b_2$, so in this subtree the root is labeled by y_2, its left subtree checks if $(0, 1) \le (y_3, y_4)$, and its right subtree checks if $(y_3, y_4) \le (1, 0)$. The DNF formula for this tree has a term for each leaf labeled by 1; the formula is

$$(y_1 \wedge \neg y_2 \wedge \neg y_3 \wedge y_4) \vee (y_1 \wedge \neg y_2 \wedge y_3) \vee (y_1 \wedge y_2 \wedge \neg y_3) \vee (y_1 \wedge \neg y_2 \wedge y_3 \wedge \neg y_4).$$

[8] A decision tree is an ordered binary tree such that each leaf is labeled by 0 or 1 and each internal node is labeled by a variable. To check if the decision tree accepts an assignment, we start at the root of the tree and traverse until we reach a leaf according to the following rule: If the assignment satisfies the variable in the internal node, we go to its right son, otherwise we go to its left son. The assignment is accepted if we reach a leaf labeled by 1 and is rejected otherwise.

To construct a decision tree checking membership in a d-dimensional rectangle specified by $a^1, \ldots, a^d$, $b^1, \ldots, b^d$, we first construct a decision tree for $a^1 \leq y^1 \leq b^1$. We then replace each leaf labeled by 1 with a decision tree checking if $a^2 \leq y^2 \leq b^2$. We repeat this process for each dimension. All together, we get a decision tree of depth $O(nd)$, which has $(O(n))^d$ leaves, thus, it can be described by a DNF formula with $(O(n))^d$ terms. Finally, if our input is a union of ℓ rectangles, we can represent it as a DNF formula with $\ell(O(n))^d$ terms, hence:

Theorem 4.15. *There exists an efficient resemblance-preserving algorithm for finding a point in a union of ℓ rectangles in $[N]^d$. The running time of the algorithm is* $\text{poly}((\log N)^d, \ell)$.

The above algorithm is polynomial in ℓ and $(\log N)^d$, while the size of the input is $O(\ell d \log N)$, thus, it is polynomial when d is constant. It would be interesting to construct an efficient algorithm for nonconstant d. Notice that a union of ℓ rectangles in $[2]^d$ is equivalent to an ℓ-term DNF formula with d variables. Thus, there is a polynomial resemblance-preserving algorithm for union of rectangles in $[2]^d$.

5. Relationships Between the Classes of Private Algorithms

Figure 5 summarizes the main classes of private algorithms discussed in this paper and their containment relationships. We now show that different search problems admit different types of private algorithms.

5.1. *Deterministic Private Algorithms: Lexicographically First vs. Median*

Claim 5.1. *There exists a polynomial time deterministic algorithm for outputting the lexicographically first assignment of DNF formulae.*

Proof sketch. We use the fact that it is possible to decide in polynomial time whether a DNF formula ϕ is satisfiable. Given a satisfiable DNF formula ϕ over variables $x_1, \ldots, x_n$, our algorithm constructs the formula $\phi' = \phi|_{x_1=0}$. If ϕ' is satisfiable, let $a_1 = 0$, and otherwise let $a_1 = 1$. Continue by finding the lexicographically first assignment for the formula $\phi|_{x_1=a_1}$. □

Claim 5.2. *Unless* RP = NP, *there is no polynomial time deterministic algorithm for outputting the median assignment of DNF formulae.*

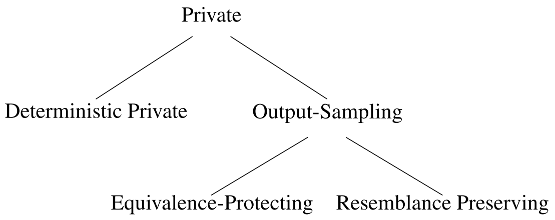

Fig. 5. The algorithm classification tree.

Proof sketch. We use the fact that (the decision problem) UNIQUE-SAT is NP-hard under randomized reductions [20] and show that it is possible to reduce UNIQUE-SAT to finding the median assignment of DNF formulae. Let ϕ be an instance of UNIQUE-SAT (i.e., ϕ is a CNF formula (over the variables $x_1, \ldots, x_n$) that is either unsatisfiable or has a unique satisfying assignment). Construct a DNF formula ψ by exchanging conjuncts with disjuncts and negating all literals. It is easy to see that an assignment $a = (a_1, \ldots, a_n)$ satisfies ϕ iff it does not satisfy ψ. Hence, if ϕ is unsatisfiable, then ψ is a tautology, and its median assignment is $(1, 0, \ldots, 0)$; otherwise there exists a unique assignment that does not satisfy ψ.

We now show how to check whether ψ is a tautology. We first compute the median satisfying assignment of ψ and note that if it is not $(1, 0, \ldots, 0)$, then ψ is not a tautology. It is, however, possible that the unique assignment that does not satisfy ψ is above $(1, 0, \ldots, 0)$ in lexicographic order. We hence continue recursively with the formula $\psi|_{x_1=1}$. □

5.2. *Deterministic Private vs. Output-Sampling Algorithms*

We now demonstrate a search problem that admits an output-sampling algorithm (recall that an output-sampling algorithm outputs a uniformly chosen solution on each instance) but no efficient deterministic private, equivalence-protecting, or resemblance-preserving algorithm.

Definition 5.3. Define

$$\mathcal{QR}(N, a) = \left\{ar^2 \bmod N : r \in \mathbb{Z}_N^*\right\}.$$

Claim 5.4. *The problem $\mathcal{QR}$ admits a polynomial time output-sampling algorithm but no efficient deterministic private, equivalence-protecting, or resemblance-preserving algorithm, unless quadratic residuosity is decidable in polynomial time.*

Proof. Construct an algorithm $\mathcal{A}_{\mathcal{QR}}$ that on input (N, a) selects a random $r \in \mathbb{Z}_N^*$ and outputs $b = r^2 \cdot a \bmod N$. It is easy to see that b is chosen uniformly from $\mathcal{QR}(N, a)$. Hence, algorithm $\mathcal{A}_{\mathcal{QR}}$ is an output-sampling algorithm.

To see the infeasibility result for the class of deterministic private algorithms, assume the existence of such an algorithm $\mathcal{A}_{\mathcal{QR}}$ for $\mathcal{QR}$ and construct a (deterministic) algorithm $\mathcal{B}$ for deciding quadratic residuosity as follows.[9] On input (N, a) algorithm $\mathcal{B}$ applies $\mathcal{A}_{\mathcal{QR}}$ on $(N, 1)$ and on (N, a). It answers "QR" if $\mathcal{A}_{\mathcal{QR}}$ outputs the same element of $\mathbb{Z}_N^*$ on both invocations and "NQR" otherwise. Note that $\mathcal{QR}(N, a) = \mathrm{QR}_N$ if and only if $a \in \mathrm{QR}_N$. Since $\mathcal{A}_{\mathcal{QR}}$ is a deterministic private algorithm (with answers depending only on the solution set), for every N, it returns the same quadratic residue for all $a \in QR_N$. The correctness follows since $1 \in \mathrm{QR}_N$ and hence $\mathcal{QR}(N, 1) = \mathrm{QR}_N$.

The argument for showing infeasibility of equivalence-protecting and resemblance-preserving algorithms is similar. We modify algorithm $\mathcal{B}$ to be a randomized algorithm

[9] The Quadratic Residuosity problem is to decide, given a modulus N and a number $a \in \mathbb{Z}_N^*$, whether $a \in \mathrm{QR}_N = \{r^2 : r \in \mathbb{Z}_N^*\}$. It is believed that deciding quadratic residuosity when $N = p \cdot q$ for large primes p, q is intractable.

that first chooses a random seed s (of the appropriate length) for $\mathcal{A}_{\mathcal{QR}}$ and then acts like the deterministic algorithm described above. □

The next example is of a search problem that admits a deterministic private algorithm but no (nontrivial) output-sampling algorithm.

Definition 5.5. For a CNF formula ϕ over Boolean variables $x_1, \ldots, x_n$, define

$$\mathcal{ZERO}\text{–}\mathcal{SAT}(\phi) = \{a \in \{0,1\}^n : a = 0^n \vee \phi(a) = \text{TRUE}\}.$$

We say that a randomized algorithm $\mathcal{A}$ for $\mathcal{ZERO}\text{–}\mathcal{SAT}$ is *nontrivial* if $\mathcal{A}(\phi)$ returns, with a non-negligible probability, a nonzero assignment whenever ϕ is satisfiable by some nonzero assignment.

Claim 5.6. *The problem $\mathcal{ZERO}\text{–}\mathcal{SAT}$ admits a deterministic polynomial time private algorithm, but, unless $RP = NP$, no nontrivial output-sampling algorithm for $\mathcal{ZERO}\text{–}\mathcal{SAT}$ exists.*

Proof. Consider the algorithm that on input a CNF formula ϕ over Boolean variables $x_1, \ldots, x_n$ outputs 0^n. This is a private deterministic algorithm for $\mathcal{ZERO}\text{–}\mathcal{SAT}$.

To get the infeasibility result, assume $\mathcal{A}_{\mathcal{ZERO}-\mathcal{SAT}}$ is a non-trivial output-sampling algorithm for the search problem $\mathcal{ZERO}\text{–}\mathcal{SAT}$, and let $p(n)$ be a polynomial such that, on satisfiable formulae over $x_1, \ldots, x_n$,

$$\Pr[\mathcal{A}_{\mathcal{ZERO}-\mathcal{SAT}}(\phi) \neq 0^n] \geq 1/p(n).$$

Construct a randomized algorithm $\mathcal{B}$ for deciding SAT. On input a CNF formula ϕ, algorithm $\mathcal{B}$ first checks if $\phi(0^n)$ is satisfied. Otherwise, it applies $\mathcal{A}_{\mathcal{ZERO}-\mathcal{SAT}}$ for a total of $n \cdot p(n)$ times. If in any of the invocations $\mathcal{A}_{\mathcal{ZERO}-\mathcal{SAT}}$ returns a nonzero assignment satisfying ϕ, then $\mathcal{B}$ outputs “YES,” otherwise it outputs “NO.” It is easy to see that algorithm $\mathcal{B}$ runs in polynomial time, accepts all satisfiable formulae with probability $1 - e^{-n}$, and rejects all non-satisfiable formulae. If $RP \neq NP$, then no such efficient algorithm $\mathcal{B}$ exists, and, thus, no efficient nontrivial randomized private algorithm for $\mathcal{ZERO}\text{–}\mathcal{SAT}$ exists. □

5.3. *Resemblance-Preserving vs. Equivalence-Protecting Algorithms*

In Sect. 4.3 we saw that a resemblance-preserving algorithm exists for the search problem of finding satisfying assignments for DNF formulae. However, no efficient equivalence-protecting algorithm for DNF exists, unless NP⊆BPP (and hence using self reduction of NP-complete problems, RP = NP) since such an algorithm can be used to check if two DNF formulae are equivalent, a problem that is coNP-complete (since it is possible to reduce DNF-TAUTOLOGY to DNF equivalence by setting one of the formulae to a tautology). This problem does exhibit efficient deterministic private algorithms as described in Sect. 5.1.

Our last example is of a search problem that has an equivalence-protecting algorithm but no resemblance-preserving algorithm. This example is a variation on the quadratic residuosity problem discussed above, where we ensure that there are no equivalent inputs, however, there are inputs with nearly identical sets of solutions.

Definition 5.7. Define

$$\mathcal{QR}^*(N,a) = \bigl\{\bigl(j, ar^2 \bmod N\bigr) : 1 \le j \le N, r \in \mathbb{Z}_N^*\bigr\} \cup \bigl\{(0,a)\bigr\}.$$

Claim 5.8. *The problem $\mathcal{QR}^*$ admits a polynomial time equivalence-protecting algorithm but no efficient resemblance-preserving algorithms, unless quadratic residuosity is decidable in polynomial time.*

Proof. Given an input (N,a), let $n = \lceil \log n \rceil$, that is, n is the number of bits in the representation of N. We construct an equivalence-protecting algorithm $\mathcal{A}_{\mathcal{QR}^*}$ that uses the seed s_n as a seed for a family $\{F_n\}_{n\in\mathbb{N}}$ of pseudorandom functions. On input (N,a), the algorithm $\mathcal{A}_{\mathcal{QR}^*}$ uses $f_{s_n}(N,a)$ to sample $r \in \mathbb{Z}_N^*$ and $j \in \{1,\dots,N\}$ and returns $(j, ar^2 \bmod N)$.

The argument for showing infeasibility of resemblance-preserving algorithms for $\mathcal{QR}^*$ is similar to the infeasibility of such an algorithm for $\mathcal{QR}$. The only difference is that for every $a \in \mathrm{QR}_N$, the sets $\mathcal{QR}^*(N,1)$ and $\mathcal{QR}^*(N,a)$ have symmetric difference two, thus their resemblance is large, and thus a resemblance-preserving algorithm must return the same answer on them with high probability. □

6. Discussion

Understanding how the choice of a solution for a search problem affects the extent that information is leaked is essential in coming up with protocols that preserve meaningful privacy. At present, no theory exists for analyzing this leakage and for matching such choices with the privacy needs of applications.

The research of private approximation and private search [1,2,11,17,18] and the results herein suggests that the picture may be quite complicated. For instance, we would like to be able to postulate all (or most) privacy concerns in a single standard definition. However, as we showed in Sect. 5 above, search problems may admit significantly different privacy requirements, and hence the decision on privacy requirements cannot be disentangled from complexity considerations.

We conclude with a few (more concrete) problems that emerge from our work:

1. Theorem 3.4 relates the construction of equivalence-protecting algorithms to finding canonical representative and output-sampling algorithms. Which search problems admit such algorithms?
2. Similarly, for which search problems it is possible to efficiently compute a minimum element under a min-wise or a pairwise permutation (hence, yielding resemblance-preserving algorithms)?
3. As we noted in Sect. 1.1.1, the Gale–Shapley algorithm [13] for stable matching is not a private search algorithm. While a deterministic private algorithm for stable marriage exists (by computing a lexicographically first solution), the question of constructing an equivalence-protecting algorithm or an output-sampling algorithm for stable marriage is still open, even in restricted scenarios [5].

Acknowledgements

Part of this research was performed when the authors visited IPAM at UCLA. We thank Rafi Ostrovsky, Mark Green, and the IPAM staff for inviting us to IPAM and making our stay pleasant and productive. We would like to thank the anonymous reviewers for their useful comments.

Appendix A. Proof of Theorem 4.9

We first prove that

$$\Pr\bigl[\pi_s(a_1)=\min \pi_s(A)\bigr] \le \frac{2}{\sqrt{|A|-1}-1}.$$

Let $A=\{a_1,a_2,\dots,a_k\}$, where $k=|A|$, and let $U=[N]$ for some integer N. For all $n_0 \in [N]$, the probability that $\pi_s(a_1)$ attains the minimum is bounded by

$$\Pr\bigl[\pi_s(a_1)=\min \pi_s(A)\bigr] \le \Pr\bigl[\pi_s(a_1)\le n_0\bigr] + \Pr\bigl[\pi_s(a_1)>n_0 \text{ for all } a_2,\dots,a_k\bigr]. \qquad (A.1)$$

Define $\alpha = n_0/N$. We will bound each of the terms in the above sum separately. For the first term, note that $\Pr[\pi_s(a_1)\le n_0]=n_0/N=\alpha$.

For $2\le i\le k$, let $B_i=1$ if $\pi_s(a_i)>n_0$ and $B_i=0$ otherwise, and let $B=\sum_{1<i\le k} B_i$. Using linearity of expectation,

$$E[B]=(k-1)\cdot\frac{N-n_0}{N}=(k-1)(1-\alpha).$$

Computing the second moment, we get that

$$E\bigl[B^2\bigr]=2\sum_{1<i<j\le k}E[B_iB_j]+\sum_{1<i\le k}E[B_i].$$

The equality follows since $B_i^2=B_i$. Using pairwise independence, we get that

$$\begin{aligned}E[B_iB_j]&=\Pr\bigl[\pi_s(a_i)>n_0\wedge \pi_s(a_j)>n_0\bigr]\\&=(N-n_0)(N-n_0-1)\cdot\frac{1}{N(N-1)}\le(1-\alpha)^2,\end{aligned}$$

and hence

$$E\bigl[B^2\bigr]\le(k-1)(k-2)(1-\alpha)^2+(k-1)(1-\alpha).$$

We get that

$$\operatorname{Var}[B]=E\bigl[B^2\bigr]-\bigl(E[B]\bigr)^2\le(k-1)\alpha(1-\alpha).$$

Using Chebyshev's inequality, we get that

$$\begin{aligned}\Pr[B \geq k-1] &\leq \Pr\left[B - E[B] \geq \sqrt{\operatorname{Var}[B]} \cdot \sqrt{\frac{(k-1)\alpha}{1-\alpha}}\right] \\ &\leq \frac{1-\alpha}{\alpha(k-1)} \leq \frac{1}{\alpha(k-1)}.\end{aligned}$$

Thus, by (A.1),

$$\Pr\bigl[\pi_s(a_1) = \min \pi_s(A)\bigr] \leq \alpha + \frac{1}{\alpha(k-1)}. \tag{A.2}$$

Choosing $n_0 = \lfloor N/\sqrt{k-1} \rfloor$, we obtain $\alpha = n_0/N = \lfloor N/\sqrt{k-1}\rfloor/N$. Substituting this value of α into (A.2), we conclude that

$$\begin{aligned}\Pr\bigl[\pi_s(a_1) = \min \pi_s(A)\bigr] &\leq \frac{N/\sqrt{k-1}}{N} + \frac{1}{(k-1)(\frac{N}{\sqrt{k-1}} - 1)/N} \\ &\leq \frac{1}{\sqrt{k-1}} + \frac{1}{\sqrt{k-1} - (k-1)/N} \\ &\leq \frac{2}{\sqrt{k-1} - 1}.\end{aligned}$$

The lower bound on $\Pr[\pi_s(a_1) = \min \pi_s(A)]$ is a simple application of the union bound. For $n_0 \in [N]$, by the pairwise independence we have $\Pr[\pi_s(a_i) < n_0 | \pi_s(a_1) = n_0] = (n_0 - 1)/(N-1)$. Hence,

$$\Pr\bigl[\pi_s(a_i) > n_0 \text{ for all } 1 < i \leq k | \pi_s(a_1) = n_0\bigr] \geq 1 - \frac{(k-1)(n_0-1)}{N-1}.$$

Noting that $1 - (k-1)(n_0-1)/(N-1)$ is nonnegative for $n_0 \leq \lfloor (N-1)/(k-1)\rfloor + 1$, we get

$$\begin{aligned}&\Pr\bigl[\pi_s(a_1) = \min \pi_s(A)\bigr] \\ &\quad= \sum_{n_0 \in [N]} \Pr\bigl[\pi_s(a_1) = n_0\bigr] \cdot \Pr\bigl[\pi_s(a_i) > n_0 \text{ for all } 1 < i \leq k | \pi_s(a_1) = n_0\bigr] \\ &\quad\geq \frac{1}{N} \sum_{n_0=1}^{\lfloor (N-1)/(k-1)\rfloor + 1} \left(1 - \frac{(k-1)(n_0-1)}{N-1}\right) \\ &\quad= \frac{1}{N} \cdot \frac{1}{2}\left(\left\lfloor \frac{N-1}{k-1} \right\rfloor + 1\right)\left(2 - \frac{(k-1)\lfloor (N-1)/(k-1)\rfloor}{N-1}\right) \\ &\quad\geq \frac{1}{2N}\left(\frac{N}{k-1}\right)\left(2 - \frac{(k-1)(N-1)/(k-1)}{N-1}\right) = \frac{1}{2(k-1)}.\end{aligned}$$

References

[1] A. Beimel, P. Carmi, K. Nissim, E. Weinreb, Private approximation of search problems, in *Proc. of the 38th ACM Symp. on the Theory of Computing*, pp. 119–128, 2006

[2] A. Beimel, R. Hallak, K. Nissim, Private approximation of clustering and vertex cover, in *Proc. of the Fourth Theory of Cryptography Conference—TCC 2007*, ed. by S. Vadhan. Lecture Notes in Computer Science, vol. 4392 (Springer, Berlin, 2007), pp. 383–403

[3] M. Ben-Or, S. Goldwasser, A. Wigderson, Completeness theorems for noncryptographic fault-tolerant distributed computations, in *Proc. of the 20th ACM Symp. on the Theory of Computing*, pp. 1–10, 1988

[4] E.R. Berlekamp, Factoring polynomials over large finite fields. *Math. Comput.* **24**, 713–735 (1970)

[5] N. Bhatnagar, S. Greenberg, D. Randall, Sampling stable marriages: Why spouse-swapping won't work, in *Proc. of the 19th ACM-SIAM Symp. on Discrete Algorithms*, pp. 1223–1232, 2008

[6] A.Z. Broder, On the resemblance and containment of documents, in *Compression and Complexity of Sequences 1997*, pp. 21–29, 1997

[7] A.Z. Broder, S.C. Glassman, M.S. Manasse, G. Zweig, Syntactic clustering of the web, in *Proc. of World Wide Web Conference*, pp. 1157–1166, 1997

[8] A.Z. Broder, M. Charikar, A.M. Frieze, M. Mitzenmacher, Min-wise independent permutations. *J. Comput. Syst. Sci.* **60**(3), 630–659 (2000)

[9] D. Chaum, C. Crépeau, I. Damgård, Multiparty unconditionally secure protocols, in *Proc. of the 20th ACM Symp. on the Theory of Computing*, pp. 11–19, 1988

[10] V.M.F. Dias, G.D. da Fonseca, C.M.H. de Figueiredo, J.L. Szwarcfiter, The stable marriage problem with restricted pairs. *Theor. Comput. Sci.* **306**(1–3), 391–405 (2003)

[11] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M.J. Strauss, R.N. Wright, Secure multiparty computation of approximations. *ACM Trans. Algorithms* **2**(3), 435–472 (2006). Conference version, in *Proc. of the 28th International Colloquium on Automata, Languages and Programming*. Lecture Notes in Computer Science, vol. 2076 (Springer, Berlin, 2001), pp. 927–938

[12] M. Franklin, M. Gondree, P. Mohassel, Improved efficiency for private stable matching, in *Topics in Cryptology – CT-RSA 2007*, ed. by M. Abe. Lecture Notes in Computer Science, vol. 4377 (Springer, Berlin, 2007), pp. 163–177

[13] D. Gale, L.S. Shapley, College admissions and the stability of marriage. *Am. Math. Mon.* **69**, 9–15 (1962)

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

[15] O. Goldreich, S. Micali, A. Wigderson, How to play any mental game, in *Proc. of the 19th ACM Symp. on the Theory of Computing*, pp. 218–229, 1987

[16] P. Golle, A private stable matching algorithm, in *10th International Conference on Financial Cryptography and Data Security*, ed. by G. Di. Lecture Notes in Computer Science, vol. 4107 (Springer, Berlin, 2006), pp. 65–80

[17] S. Halevi, R. Krauthgamer, E. Kushilevitz, K. Nissim, Private approximation of NP-hard functions, in *Proc. of the 33th ACM Symp. on the Theory of Computing*, pp. 550–559, 2001

[18] P. Indyk, D. Woodruff, Polylogarithmic private approximations and efficient matching, in *Proc. of the Third Theory of Cryptography Conference—TCC 2006*, ed. by S. Halevi, T. Rabin. Lecture Notes in Computer Science, vol. 3876 (Springer, Berlin, 2006), pp. 245–264

[19] M. Jerrum, A. Sinclair, E. Vigoda, A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. ACM* **51**(4), 671–697 (2004)

[20] L.G. Valiant, V.V. Vazirani, NP is as easy as detecting unique solutions. *Theor. Comput. Sci.* **47**, 85–93 (1986)

[21] A.C. Yao, Protocols for secure computations, in *Proc. of the 23th IEEE Symp. on Foundations of Computer Science*, pp. 160–164, 1982