How Should We Solve Search Problems Privately?

Amos Beimel[1], Tal Malkin[2⋆], Kobbi Nissim[1⋆⋆], and Enav Weinreb[3]

[1] Dept. of Computer Science, Ben-Gurion University. Be'er Sheva, Israel.
{beimel|kobbi}@cs.bgu.ac.il
[2] Dept. of Computer Science, Columbia University, New York, NY.
tal@cs.columbia.edu
[3] Dept. of Computer Science, Technion, Haifa, Israel.
weinreb@cs.technion.ac.il

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 to a set of linear equations.
Picking one output arbitrarily from the solution set has significant implications on the privacy of the algorithm. Beimel et al. [STOC 2006] gave a *minimal* definition for private computation of search problems with focus on proving impossibility result. 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.

1 Introduction

Secure multiparty computation addresses a setting where several distrusting parties want to jointly compute a function $f(x_1, \ldots, x_n)$ of their private inputs $x_1, \ldots, x_n$, while maintaining the privacy of their inputs. One of the most fundamental, and by now well known, achievements in cryptography (initiated by [19, 14, 8, 3], and continued by a long line of research) shows that in fact for any feasible function f, there exists a secure multiparty protocol 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 instance $x = (x_1, \ldots, x_n)$. For example, the parties may jointly hold a graph and wish to compute a shortest path between two of

⋆ Research partially supported by the NSF (grant No. CCF-0347839).
⋆⋆ Research partially supported by the Israel Science Foundation (grant No. 860/06).

its vertices or to find a minimal vertex cover in it.[4] We call such problems *search problems*. In such cases, to apply known results of secure multiparty computation, one has first to decide upon a polynomial-time computable function that solves the search problem.

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 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, and it is clear that if the computed function leaks unnecessary 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 answering the client's queries in a way that reveals the least information possible on the database. However, the strategy the server chooses to answer each query might reveal information. For example, consider a case where the client queries for the name of a person whose details are in the database and satisfies some condition. An arbitrary solution such as answering with the details of the appropriate person whose name is the lexicographically first in the database reveals the fact that every person prior to that person in the lexicographic order does not satisfy the given condition.

In this paper we study the privacy implications of how the output is chosen for search problems, propose suitable privacy requirements, and provide constructions achieving them for several problems (see details below). This generalizes the approach of [1], who introduced the problem of private search algorithms in the context of private approximations. Beimel et al. [1] have put forward what seems to be a minimal requirement of privacy (first coined in the context of private approximation of functions [10], and later extended to search problems):

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

That is, in order for the algorithm to be private, the output must depend only on the solution set, and not on the specific input. In spirit of this requirement, we say that two inputs are equivalent if they have the same set of solutions.

This definition was reasonable in the context of [1] because they provide mostly negative results (so a weaker definition corresponds to stronger infeasibility results), and because in the context of private approximations of func-

[4] Another example is when the parties compute an approximation to a function $f()$ as in [10, 16, 17]. Again, there is potentially more than one correct answer for an instance.

tions[5], this turns out to be a significant privacy guarantee (as it implies that no information beyond the original $f(x)$ is leaked). However, in the context of search problems the implication is potentially much weaker – that no information beyond the *entire solution set* of x is leaked. Arguably, for most applications requiring privacy, leaking information up to the entire solution set does not provide a sufficient privacy guarantee. Furthermore, even with this minimal definition of privacy, the notion of private search has so far proved to be very problematic. Search versions of many NP-complete problems do not admit even very weakened notion of private approximation algorithms [1, 2], and private search is infeasible even to some problems that do admit polynomial time search algorithms.

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

1.1 This Work

As discussed above, the outcome of a private algorithm $\mathcal{A}$ when run on an instance x should only depend on the set of possible solutions to x. Which further requirements should be imposed on this outcome? While the answer to this question may be application dependent, we identify two (incomparable) requirements that are suitable in many situations, 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 that are used to demonstrate essential privacy considerations arising for search problems, and to facilitate our actual proposed definitions and algorithms.

Deterministic vs. Randomized Private Algorithms. Consider first requiring any private algorithm $\mathcal{A}$ to be *deterministic*. As such, it consistently selects one of the solutions, hence subsequent applications 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 [9]).[6] Deterministic algorithms, however, leak definite information, that (depending on the application) may turn to be crucial. E.g., the lexicographically first solution rules out all solutions that are ordered below it. Furthermore, deterministic algorithms would enable verifying that the instance x is not equivalent to another instance y, even if x, y have

[5] And similarly for those instances of search problems for which a *unique* solution exists.

[6] The recent protocols of [15, 11] also output a deterministic solution – the outcome of the Gale-Shapley algorithm [12]. However, this is not a private search algorithm.

similar solution sets, just by checking the outcome of the algorithm on both instances.

Next, consider a *randomized* algorithm $\mathcal{A}$, which on input x selects from the set of solutions according to a specific distribution (depending only 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, as the information they leak is potentially "blurred". For example, if instances x, y have similar solution sets, 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. E.g., for the problem of finding a solution for a linear system of equations, the number of revealed solutions grows 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 a uniformly selected solutions exist, but no deterministic private algorithm exists (under standard assumptions), and vice versa (see Appendix A).

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 are less desirable than algorithms that do not need to maintain any state information.[7] However, we note that the state information of seeded algorithms is rather easy to maintain, as they 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.

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 (in analogy to [10, 16, 1, 2]). 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) directly 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 [19, 14, 8, 3]. In the general setup of

[7] 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.

distributed multiparty computation, however, one may also consider definitions that allow leakage to a party of any information implied by its individual input.

Equivalence Protecting Algorithms. 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.[8]

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.

In Section 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 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 a directed graph. 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 non-equivalent 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 for non-equivalent instances are highly correlated such that pairs of instances that have close output sets are answered identically with high probability.

In Section 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 P=NP.

To summarize, we present two definitions (suitable for different applications), provide technical tools to achieve these definitions, and identify generic classes,

[8] Such a random oracle can be thought of as an ideal model solution to the problem, which this definition requires to emulate.

as well as specific examples, of search problems where our tools can be used to yield private search algorithms with the desired properties. The main conceptual contribution of the paper is in putting forward the need to study private computation of search problems (where a non-private solution is well known), analyzing privacy considerations, and defining equivalence protecting and resemblance preserving algorithms. The main technical contribution of the paper is in the tools and algorithms presented in Section 4 for resemblance preserving algorithms.

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 1 (Search Problem). *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. *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 recall the *minimal* definition of private search algorithms from [1]. All our definitions will be stronger – an algorithm that satisfies Definition 7 or Definition 13 trivially satisfies Definition 3.

Definition 3 (Private Search Algorithms [1]). *A probabilistic polynomial time algorithm $\mathcal{A}_{\mathcal{P}}$ is a private search 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) for every polynomial-time algorithm $\mathcal{D}$ and for every positive polynomial $q(\cdot)$, there exists some $n_0 \in \mathbb{N}$ such that for every $x, y \in \{0,1\}^*$ such that $x \equiv_{\mathcal{P}} y$ 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 $x \equiv_{\mathcal{P}} y$, every polynomial time algorithm $\mathcal{D}$ cannot distinguish if the input of $\mathcal{A}_{\mathcal{P}}$ is x or y.

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

Definition 4 (Pseudorandom Functions [13]). *A function ensemble $F = \{F_n\}_{n\in\mathbb{N}}$ of functions from $\{0,1\}^n$ to $\{0,1\}^{\ell(n)}$ is called* pseudorandom *if for every probabilistic polynomial time oracle machine M, every polynomial $p(\cdot)$, and all sufficiently large n's,*

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

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

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

Definition 5 (Seeded Algorithms). *A seeded algorithm $\mathcal{A}$ is a deterministic polynomial time algorithm taking two inputs x, 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 for every choice of the seed s_n, i.e., $\mathcal{A}(x, s_n) = \mathcal{A}(y, s_n)$ for all $s_n \in \{0,1\}^{p(|x|)}$ whenever $x \equiv_{\mathcal{P}} y$.

3 Equivalence Protecting Privacy Definition

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 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 6 (Private Oracle). *Let $\mathcal{P} = \{\mathcal{P}_n\}_{n\in\mathbb{N}}$ be a search problem and p be the polynomial such that $\mathcal{P}_n : \{0,1\}^n \to 2^{\{0,1\}^{p(n)}}$. We say that for a given $n \in \mathbb{N}$ an oracle $O_n : \{0,1\}^n \to \{0,1\}^{p(n)}$ 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 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 7 (Equivalence Protecting Algorithm). *Let $\mathcal{P} = \{\mathcal{P}_n\}_{n\in\mathbb{N}}$ be a search problem. An algorithm $\mathcal{A}(\cdot,\cdot)$ is private with respect to $\equiv_{\mathcal{P}}$, if for every polynomial time oracle machine $\mathcal{D}$, for every polynomial p, and for all sufficiently large n's,*

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

where the first probability is over the uniform distribution over oracles O_n that are private with respect to $\mathcal{P}$, and the second probability is uniform 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. We note that, for some applications, other distributions might be preferred; the definition can be easily adjusted to such scenarios. We note that using the uniform distribution is common in many sampling algorithms, e.g., [18].

The following two definitions will be helpful in constructing equivalence protecting algorithms for various search problems. The first definition discusses algorithms that return a representative element for every equivalence class of the search problem $\mathcal{P}$. The second defines sampling an answer from the output set of a given input.

Definition 8 (Canonical Representative Algorithm). *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) for every $x \in \{0,1\}^n$ it holds that $x \equiv_{\mathcal{P}} \mathcal{A}(x)$; and (ii) for every $x, y \in \{0,1\}^n$, it holds that $\mathcal{A}(x) = \mathcal{A}(y)$ iff $x \equiv_{\mathcal{P}} y$.*

Definition 9 (Output Sampling Algorithm). *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 $\mathbf{Unif}_{\mathcal{P}(x)}$, the uniform distribution on the possible outputs on x.*

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 on an input $x \in \{0,1\}^n$. The proof of Theorem 1 is omitted here.

`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 $\mathrm{sol} \in \mathcal{P}_n(x)$.

1. Compute $y = \mathcal{A}_{\mathrm{rep}}(x)$.
2. Compute $r = F_{s_n}(y)$.
3. Output $\mathrm{sol} = \mathcal{A}_{\mathrm{rand}}(y, r)$.

Theorem 1. *Let $\mathcal{P}$ be a search problem. Suppose $\mathcal{P}$ has (i) an efficient output sampling algorithm $\mathcal{A}_{\mathrm{rand}}$; and (ii) an efficient canonical representative algorithm $\mathcal{A}_{\mathrm{rep}}$. Then Algorithm* `General Equivalence Protecting` *is an efficient equivalence protecting algorithm for $\mathcal{P}$.*

3.1 Private Algorithms for Monotone Search problems

In view of Theorem 1, the construction of a private algorithm for a given search problem is reduced to finding a canonical representative algorithm and an output sampling algorithm.We focus on search problems in which an output is a subset of the input 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 10 (Monotone Search Problem). *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 outputs of X are the solutions that are contained in X.*

For example, the problem of finding a perfect matching in a bipartite graph is monotone. The global set of solution 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 perfect matching graphs whose edges are contained in G.

Definition 11 (Relevant Element). *Let $\mathcal{P}$ be a subset search problem and X 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. The following claim shows that computing $R(X)$ efficiently from X is sufficient to get a representation algorithm.

Claim 1. *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 $Y \subseteq X$ and 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. As $X \equiv_P Y$, 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 $\equiv_P$. □

3.2 Applications of the Construction

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

Example 1 (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. [18]. 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 a output sampling algorithm for perfect matching, and thus by Theorem 1, we get that perfect matching has an efficient equivalence protecting algorithm.

Example 2 (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 $My = v$. As 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. Elementary linear algebra argument shows that if two systems have the same sets of solutions, then they have the same structure after performing the Gaussian elimination procedure. 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 (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, the set of edges that appear in any shortest path in G still form an appropriate solution for the canonical representative algorithm. Checking whether an edge is relevant for G is an easy tasks. To sample a random solution do: (i) Compute for every $v \in V$ the number of shortest paths from v 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 1, shortest path has an efficient equivalence protecting algorithm.

Similar ideas are applicable for finding shortest path in a weighted directed graph. Here, however, we do not apply Theorem 1 directly. The equivalence protecting algorithm in this case does the following: (i) Compute the set of edges that appear in at least one shortest path from s to t. (ii) Output a *random* path from s to t in the non-weighted graph computed in (i) (not a shortest path!). The randomness for step (ii) should be extracted like in Theorem 1, by applying a pseudorandom function on the graph computed in stage (i). This example is different in the fact that the canonical input we use in step (ii) is an instance to 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 Section 3. The motivation for the definition in this section is that we want the output of the algorithm will 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 be satisfied by a deterministic algorithm for non-trivial search problems (the algorithm would have to output the same "solution" for all inputs contradicting the correctness of the 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 notion used in [5, 7, 6] and seems to capture well the informal notion of "roughly the same." For example, in [5, 7] resemblance between documents was successfully used for clustering documents.

Definition 12 (Resemblance). *Let U be a set, and $A, B \subseteq U$. Then the resemblance between A and B is defined to be*

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

For a search problem $\mathcal{P}$ we will consider the resemblance between solution sets of $\mathcal{P}_n(x), \mathcal{P}_n(y)$ 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 $\mathcal{P}_n(x), \mathcal{P}_n(y)$.

Definition 13 (Resemblance Preserving Algorithm). *An algorithm $\mathcal{A}(\cdot,\cdot)$ is resemblance preserving with respect to $\mathcal{P}$ if:*

1. *For every 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 $|x| > n_0$*

$$\Big|\Pr[\mathcal{D}(x, \mathcal{A}(s_n, x)) = 1] - \Pr[\mathcal{D}(x, \mathbf{Unif}(\mathcal{P}_n(x))) = 1]\Big| \leq \frac{1}{p(|x|)} .$$

   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 outputs 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 $x, y \in \{0,1\}^*$ such that $|x| = |y|$*

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

   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 $\mathcal{P}$.*

Unlike Definition 9, in the definition of resemblance preserving algorithms we do not know how to formulate this privacy using an "ideal world". This difference implies, in particular, that in designing resemblance preserving algorithms we do not need cryptographic assumptions. In our constructions, for example, we only use pairwise independent permutations. Furthermore, Definition 13 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 (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 in 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 $\mathbb{F}_{2^n}$. As 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, as 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, that (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 compute the minimum on any set of solutions.

Definition 14 (Family of Min-wise Independent Permutations [6]). *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 1 ([6]). 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. 1, 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 Obseration 1 it is easy to see that Algorithm $\texttt{Minwise}_{\mathcal{P}}$ is perfectly resemblance preserving.

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

1. Algorithm $\texttt{Minwise}_{\mathcal{P}}$ uses a family of min-wise independent permutations. It was shown in [6] 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.[9]
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.

Algorithm $\texttt{Minwise}_{\mathcal{P}}$

INPUT: An instance $x \in \{0,1\}^*$, 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 $\mathrm{sol} \in \mathcal{P}_n(x)$.

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

Fig. 1. Algorithm $\texttt{Minwise}_{\mathcal{P}}$.

[9] We need the family of pseudorandom permutations to be secure against a non-uniform adversary. Thus, for every long enough inputs x and y a pseudorandom permutation must be min-wise. We omit further details as this is not the approach taken in this study.

Example 5 (Roots of a Polynomial). As an example for when Algorithm $\texttt{Minwise}_{\mathcal{P}}$ can be implemented efficiently we consider the problem of finding roots of a polynomial. As in Example 4, 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. Furthermore, as 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 roots, and find for which root $\pi_s(y)$ obtains a minimum. The above algorithm can be generalized to any search problems whose entire set of solutions can be generated efficiently.

Observation 2. 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 in 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 $\mathcal{P}$.

4.2 Resemblance Preserving using Pairwise Independence

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

Definition 15 (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*

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

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

Theorem 2 ([6]). *Let $\mathcal{F}$ be a family of pairwise independent permutations $\{\pi_s\}_{s\in S}$ where $\pi_s : U \to U$. Then for every set $A \subseteq U$ and every $a \in A$*

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

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

Lemma 1. *Let $\mathcal{F}$ be a family of pairwise independent permutations $\{\pi_s\}_{s\in S}$ where $\pi_s : U \to U$. Then for every sets $A, B \subseteq U$*

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

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

We construct an algorithm $\texttt{Pairwise}_{\mathcal{P}}$ that is almost identical to $\texttt{Minwise}_{\mathcal{P}}$ of Fig. 1, where the family of min-wise permutations is replaced with a family of pairwise independent permutations. The following corollary follows directly from Lemma 1:

Corollary 1. *Algorithm* $\texttt{Pairwise}_{\mathcal{P}}$ *is resemblance preserving.*

4.3 Applications of the Pairwise Independence Construction

We next show how to apply Algorithm $\texttt{Pairwise}_{\mathcal{P}}$ 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 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:

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

Linear Algebra. We show how to construct a resemblance preserving algorithm for finding a solution of a system of equations (as considered in Example 2 for Equivalence Protecting Algorithms).

Linear Algebra over $\mathbb{F}_2$. We assume that the system is over $\mathbb{F}_2$.[10] 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 $My = v$. We apply Algorithm $\texttt{Pairwise}_{\mathcal{P}}$ 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 $z = Hy + b$ for y satisfying $Ay = b$. We view $Ay = b$ and $z = Hy + v$ as a single system of linear equations with $2n$ unknowns, namely, $y = \langle y_1, \ldots, y_n \rangle$ and $z = \langle z_1, \ldots, z_n \rangle$. 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 $z_1, \ldots, z_{i-1}$ 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.

[10] In the full version of this paper we generalize the result to every finite field.

Union of Systems of Equations. 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 to 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 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 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 equation 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 $Hy_j + b = \min\{Hy_i + b : 1 \leq i \leq \ell\}$.

Theorem 3. *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.*

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 Theorem 3 and the following observations. First, the set of satisfying assignments of a single term is the set of solutions to 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 a 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 3, we get a resemblance preserving algorithm for finding a satisfying assignment of a DNF formula.

It is interesting to note that, unless P=NP, there is no efficient equivalence protecting algorithm for DNF as an equivalence protecting algorithm for DNF can be used to check if two DNF formulae are equivalent, a problem that is coNP-hard.

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 represents a rectangle as follows: First, for two points a, b we define the segment $I_{a,b} \stackrel{\text{def}}{=} \{y \in \mathbb{N} : a \leq y \leq b\}$. Second, we define $R_{a^1,\ldots,a^d,b^1,\ldots,b^d} \stackrel{\text{def}}{=} I_{a^1,b^1} \times I_{a^2,b^2} \times \cdots \times I_{a^d,b^d}$. Let $n \stackrel{\text{def}}{=} \lceil \log N \rceil$, and 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 representing each rectangle as a union of polynomially many systems of equations over $\mathbb{F}_2$, and then use Theorem 3 to construct the resemblance preserving algorithm.

Let us start with the simple case where $d = 1$ and $b^1 = \langle 1, \ldots, 1 \rangle$ (in words, $b \in \{0,1\}^n$ is the all 1 string). That is, an input is a string a and a solution is a string $y \geq a$.

$$y \geq a \text{ iff } \left(\exists_{i \in [n]} \ (y_i = 1 \wedge a_i = 0) \wedge (\forall_{1 \leq j < i} \ y_i = a_i)\right) \ \vee \ (\forall_{1 \leq j \leq n} \ (y_i = a_i)) \,. \quad (1)$$

For example, $y = \langle y_1, y_2, y_3 \rangle \geq \langle 0, 1, 0 \rangle$ either if $(y_1 = 1)$, or $(y_1 = 0 \wedge y_2 = 1 \wedge y_3 = 1)$, or $(y_1 = 0 \wedge y_2 = 1 \wedge y_3 = 0)$.

Note that, by (1), the set of points $y \geq a$ is a union of solutions of at most $n+1$ systems of equations. Similarly, the set of points $a \leq y \leq b$ is a union of solutions of at most $2(n+1)$ systems of equations: Let $a < b$ and i_0 be the minimal index such that $a_i = 0$ and $b_i = 1$ (in particular, $a_j = b_j$ for every $1 \leq j \leq i_0 - 1$).

$$a \leq y \leq b \text{ iff } (\forall_{1 \leq j < i_0} y_j = a_j) \wedge ((y_{i_0} = a_{i_0} \wedge a \leq y) \vee (y_{i_0} = b_{i_0} \wedge y \leq b)) \,. \quad (2)$$

In other words, we partitioned the segment $I_{a,b}$ to at most $2(n+1)$ segments such that the points in each segment are exactly the solutions of a system of linear equations.

Given a rectangle in $(\{0,1\}^n)^d$, we partition it to $(O(n))^d$ rectangles such that the points in each rectangle correspond to solutions of a system of linear equations, and use Theorem 3 to construct the resemblance preserving algorithm. Notice that given a rectangle $R_{a^1,\ldots,a^d,b^1,\ldots,b^d}$, we can partition each segment I_{a_i,b_i} into $O(n)$ segments $I_{i,1}, \ldots, I_{i,O(n)}$ as in (1) and (2). Thus,

$$\begin{aligned} R_{a^1,\ldots,a^d,b^1,\ldots,b^d} &= I_{a_1,b_1} \times I_{a_2,b_2} \times \cdots \times I_{a_d,b_d} \\ &= (\cup_{j_1} I_{1,j_1}) \times (\cup_{j_2} I_{2,j_2}) \times \cdots \times (\cup_{j_d} I_{d,j_d}) \\ &= \cup_{j_1,\ldots,j_d} I_{1,j_1} \times I_{2,j_2} \times \cdots \times I_{d,j_d} \,. \end{aligned}$$

Notice that for $i_1 \neq i_2$, the variables of the equations representing $I_{i_1,j_{i_1}}$ and $I_{i_2,j_{i_2}}$ are disjoint, and the points in each rectangle $I_{1,j_1} \times I_{2,j_2} \times \cdots \times I_{d,j_d}$ are solutions to a system of linear equations.

Finally, if our input is a union of ℓ rectangles, we can represent it as a union of $\ell(O(n))^d$ systems of equations, hence:

Theorem 4. *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* $\mathrm{poly}((\log N)^d, \ell)$.

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 non-constant d. Notice that a union of ℓ rectangles in $[2]^d$ is equivalent to an ℓ-term DNF formula with n variables. Thus, there is a polynomial resemblance preserving algorithm for union of rectangles in $[2]^d$.

Acknowledgments. We thank the anonymous CYRPTO referees for their useful comments. Part of this research was performed when the authors visited IPAM at UCLA. We thank Rafi Ostrovsky and the IPAM staff for inviting us to IPAM and making our stay pleasant and productive.

References

1. A. Beimel, P. Carmi, K. Nissim, and E. Weinreb. Private approximation of search problems. In *Proc. of the 38th Symp. on the Theory of Comp.*, pages 119–128, 2006.
2. A. Beimel, R. Hallak, and K. Nissim. Private approximation of clustering and vertex cover. In S. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 383–403. 2007.
3. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computations. In *Proc. of the 20th Symp. on the Theory of Comp.*, pages 1–10, 1988.
4. E. R Berlekamp. Factoring polynomials over large finite fields. *Math. Comp.*, 24:713–735, 1970.
5. A. Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997*, pages 21–29, 1997.
6. A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *J. of Computer and System Sciences*, 60(3):630–659, 2000.
7. A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. In *In Proc. of World Wide Web conference*, pages 1157 – 1166, 1997.
8. D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proc. of the 20th Symp. on the Theory of Comp.*, pages 11–19, 1988.
9. V. M. F. Dias, G. D. da Fonseca, C. M. H. de Figueiredo, and J. L. Szwarcfiter. The stable marriage problem with restricted pairs. *Theoretical Computer Science*, 306(1–3):391–405, 2003.

10. J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. J. Strauss, and R. N. Wright. Secure multiparty computation of approximations. *ACM Trans. Algorithms*, 2(3):435–472, 2006. Conference version in *Proc. of the 28th ICALP*, volume 2076 of *LNCS*, pages 927–938. 2001.
11. M. Franklin, M. Gondree, and P. Mohassel. Improved efficiency for private stable matching. In M. Abe, editor, *Topics in Cryptology – CT-RSA 2007*, volume 4377 of *LNCS*, pages 163–177. 2007.
12. D. Gale and L. S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.
13. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. of the ACM*, 33(4):792–807, 1986.
14. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proc. of the 19th Symp. on the Theory of Comp.*, pages 218–229, 1987.
15. P. Golle. A private stable matching algorithm. In G. Di Crescenzo and A. Rubin, editors, *10th International Conference on Financial Cryptography and Data Security*, volume 4107 of *LNCS*, pages 65–80. 2006.
16. S. Halevi, R. Krauthgamer, E. Kushilevitz, and K. Nissim. Private approximation of NP-hard functions. In *Proc. of the 33th Symp. on the Theory of Comp.*, pages 550–559, 2001.
17. P. Indyk and D. Woodruff. Polylogarithmic private approximations and efficient matching. In S. Halevi and T. Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 245–264. 2006.
18. M. Jerrum, A. Sinclair, and E. Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. of the ACM*, 51(4):671–697, 2004.
19. A. C. Yao. Protocols for secure computations. In *Proc. of the 23th IEEE Symp. on Foundations of Computer Science*, pages 160–164, 1982.

A Deterministic vs. Randomized Private Algorithms

We start with a search problem that admits a randomized private algorithm (outputting a uniformly chosen solution on each instance), but no efficient deterministic one. For $n = p \cdot q$ and $a \in Z_n^*$ with Jacobi Symbol $(\frac{a}{n}) = 1$ define $\mathcal{QR}(n,a) = \{b \in Z_n^* : (\frac{b}{n}) = 1 \ \ \wedge \ \ b \in QR_n \Leftrightarrow a \in QR_n\}$.

Claim 2. *The problem $\mathcal{QR}$ admits a randomized polynomial time private algorithm, but no efficient deterministic private algorithms, unless quadratic residuosity is decidable in deterministic polynomial time.*

Our second example is of a search problem that admits a deterministic private algorithm but no (non trivial) randomized one.

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

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

If a randomized algorithm for $\mathcal{ZERO} - \mathcal{SAT}$ assigns non-negligible probability to some non-zero assignment whenever ϕ is satisfiable we say it is *non-trivial.*

Claim 3. *The problem $\mathcal{ZERO} - \mathcal{SAT}$ admits a deterministic polynomial time private algorithm, but, unless $NP \subseteq RP$ no non-trivial randomized private algorithm for $\mathcal{ZERO} - \mathcal{SAT}$ exists.*