Task-Tuning in Privacy-Preserving Crowdsourcing Platforms

Joris Duguépéroux
Univ Rennes, CNRS, IRISA
Rennes, France
joris.dugueperoux@irisa.fr

Antonin Voyez
Univ Rennes, CNRS, IRISA
Rennes, France
antonin.voyez@irisa.fr

Tristan Allard
Univ Rennes, CNRS, IRISA
Rennes, France
tristan.allard@irisa.fr

ABSTRACT

Specialized worker profiles of crowdsourcing platforms may contain a large amount of identifying and possibly sensitive personal information (e.g., personal preferences, skills, available slots, available devices) raising strong privacy concerns. This led to the design of privacy-preserving crowdsourcing platforms, that aim at enabling efficient crowdsourcing processes while providing strong privacy guarantees even when the platform is not fully trusted. We propose a demonstration of the PKD algorithm, a privacy-preserving space partitioning algorithm dedicated to enabling secondary usages of worker profiles within privacy-preserving crowdsourcing platforms by combining differentially private perturbation with additively-homomorphic encryption. The demonstration scenario showcases the PKD algorithm by illustrating its use for enabling requesters tune their tasks according to the actual distribution of worker profiles while providing sound privacy guarantees.

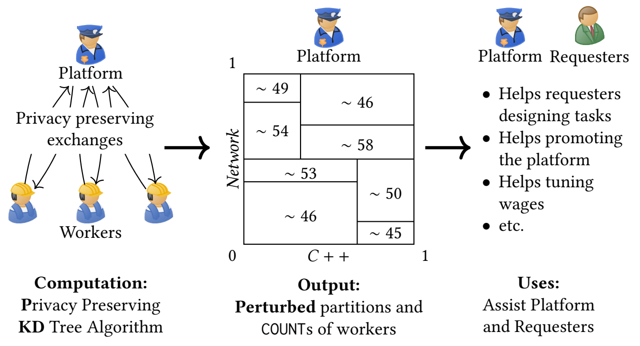


Figure 1: Overview of the PKD algorithm: supporting secondary usages of worker profiles with privacy guarantees

1 INTRODUCTION

Crowdsourcing platforms are online intermediates between requesters and workers: workers have skills and look for tasks, while requesters propose tasks that require specific skills. Crowdsourcing platforms are used in various application domains such as micro-tasks[1] or specialized software engineering[2]. Their efficiency, either for matching tasks to profiles (the primary usage of profiles) or for giving to requesters insights about the distribution of skills available within the population in order, e.g., to attract new requesters or to let requesters fine-tune their tasks according to the actual population of workers[3] (secondary usage of profiles), depends especially on the detailed information contained within worker profiles. A profile may indeed contain an arbitrary amount of information: professional or personal skills, daily availabilities, minimum wages, diplomas, professional experiences, centers of interest and personal preferences, devices owned and available, *etc.*

However fine grain worker profiles can be highly identifying or sensitive and privacy scandals have shown that those platforms are not immune to negligence or misbehaviours[4]. In a context where users expect crowdsourcing platforms to protect their personal data [Xia et al. 2017] and laws firmly require businesses and public organizations to safeguard the privacy of individuals (such as the European GDPR[5] or the California Consumer Privacy Act[6]) , designing and implementing sound privacy-preserving crowdsourcing processes is of utmost importance.

In this demonstration, we present the PKD algorithm [Duguépéroux and Allard 2019], a privacy-preserving space partitioning algorithm dedicated to enabling a wide range of secondary usages of worker profiles within privacy-preserving crowdsourcing platforms (see Figure 1). The PKD algorithm is distributed between a set of distrustful workers and an untrusted platform and builds on differentially private perturbation and additively-homomorphic encryption in order to compute a hierarchical partitioning of the skills of workers together with the approximate number of workers per partition. No raw worker profile is ever communicated to any other participant during the computation. The output of the PKD algorithm can be used for computing multidimensional COUNTs over worker profiles. The security of the PKD algorithm relies on composable security models in order to integrate well with privacy-preserving solutions to primary usages [Béziaud et al. 2017; Kajino 2015] without jeopardizing the privacy guarantees.

The demonstration scenario showcases the use of the PKD algorithm for letting requesters tune their tasks according to the actual population of workers, all this with sound privacy guarantees. The demonstration scenario essentially illustrates the impact of knowing the distribution of workers when tuning a task, and sheds the light on the tradeoff between privacy and utility within privacy-preserving crowdsourcing platforms, showing that a high privacy level can be guaranteed while still allowing high-standard secondary usages.

[1]https://www.mturk.com
[2]https://tara.ai
[3]For example, if functional language gurus are rare, it might be worth awarding more money for the task or updating the task such that it fits more common profiles.
[4]See, e.g., [Lease et al. 2013] or https://www.theverge.com/2014/11/19/7245447/uber-allegedly-tracked-journalist-with-internal-tool-called-god-view
[5]https://eur-lex.europa.eu/eli/reg/2016/679/oj
[6]https://www.caprivacy.org/

2 PRIVACY-PRESERVING INFORMED TASK-TUNING: AN OVERVIEW

2.1 Preliminaries

2.1.1 Participants. Three types of participants collaborate together during our crowdsourcing process. Workers are interested in solving tasks relevant to their profile, requesters propose tasks to be solved by appropriate workers, and the platform supports the intermediation. The number of skills n is fixed. A worker profile $p \in P$ is represented by an n-dimensional vector of floats where each value $p[j] \in [0,1]$ represents the degree of competency of the profile p with respect to skill j. A task $t \in T$ is made of two parts. The first part is the meta-data containing the requirements needed to perform the task. We model it as an n-dimensional vector of ranges over skills (*i.e.,* a subspace of the space of profiles). The second part is the detailed task description provided by the requester (an arbitrary bitstring). In this work, we focus on the metadata part.

We assume that the participants are equipped with today's commodity hardware (i.e., the typical CPU/bandwidth/storage resources of a personal computer). However, we expect the platform to be available 24/7, similarly to a traditional client/server setting.

2.1.2 Security. We assume that all participants follow the *honest-but-curious* attack model: they may use any information disclosed along the algorithm to infer information about profiles, but they do not step outside the protocol.

As stated in Definition 1, the privacy model satisfied by the PKD algorithm is a computational variant of the well-known *differential privacy* model [Dwork 2006] called ϵ_κ-SIM-CDP [Mironov et al. 2009] (see the proofs in the technical report [Duguépéroux and Allard 2019]).

Definition 1 (ϵ_κ-SIM-CDP privacy [Mironov et al. 2009] (simplified)). *The randomized function f_κ provides ϵ_κ-SIM-CDP if there exists a function F_κ that satisfies ϵ-differential privacy and a negligible function $negl(\cdot)$, such that for every set of worker profiles $\mathcal{P}$, every probabilistic polynomial time adversary A_κ, every auxiliary background knowledge $\zeta_\kappa \in \{0,1\}^*$, it holds that:*

$$|\Pr[\mathsf{A}_k(\mathsf{f}_\kappa(\mathcal{P},\zeta_\kappa)) = 1] - \Pr[\mathsf{A}_k(\mathsf{F}_\kappa(\mathcal{P},\zeta_\kappa)) = 1]| \leq negl(\kappa)$$

The original ϵ-differential privacy model applies to a randomized function f and aims at hiding the impact of any possible individual value on the possible outputs of f, often by adding random noise to it. Computational variants of differential privacy are especially relevant when differentially private perturbation and semantically secure encryption are used jointly, as done within the PKD algorithm. First, the differentially private perturbation scheme used by the PKD algorithm is the Geometric mechanism [Ghosh et al. 2012]. It consists essentially in sampling a two-sided geometric distribution parameterized by the differential privacy parameter ϵ and by the aggregate to be perturbed. It benefits from the following nice properties: it is designed for perturbing integers, and the sampling can be easily distributed over workers (infinite divisibility of the two-sided geometric distribution [Duguépéroux and Allard 2019]). Second, the PKD algorithm makes use of an *additively homomorphic encryption scheme.* Additively-homomorphic encryption schemes essentially allow to perform addition operations over encrypted data. Any additively-homomorphic encryption scheme fits our approach as long as it provides *semantic security guarantees* (usual security guarantees), *additively-homomorphic encryption* (possibility to perform additively homomorphic sums) and *non-interactive threshold decryption* (allows the decryption key to be split in K key-shares, such that a complete decryption requires to perform independently $T \leq K$ partial decryption by distinct key-shares). The Damgard-Jurik cryptosystem [Damgård and Jurik 2001], a generalization of Paillier [Paillier 1999], is an instance of encryption scheme that provides the desired properties. We refer the interested reader to the original paper for details [Damgård and Jurik 2001].

2.1.3 Quality. Roughly speaking, we evaluate the quality achieved by the outputs of the PKD algorithm by measuring the average absolute error between (approximate) counts estimated through its outputs and the corresponding exact counts computed on the raw, non-protected, worker profiles. We refer the interested reader to the technical report [Duguépéroux and Allard 2019] for more details on the quality definition and on the experimental results.

2.2 The PKD algorithm

The PKD algorithm is an adaptation of the well-known centralized KD-Tree construction algorithm [Bentley 1975], to a context where no central server is trusted, and the result is private. The resulting KD-Tree is used to estimate the underlying *multidimensional* distribution of workers (e.g., to let requesters tune their tasks accordingly). Each worker holds his profile locally and engages with the platform and other workers in the execution of the PKD algorithm. The PKD algorithm consists essentially in splitting recursively the space of skills in two and stops when a termination criterion is met (e.g., fixed number of splits). The split is performed by choosing one dimension d at each iteration (e.g., considering on dimension after the other), projecting the set of skills on d, and forming two partitions around the median. The PKD algorithm outputs a binary tree where each node is a partition of the space of skills with the (perturbed) number of worker profiles it contains. The key operation of the PKD algorithm is the distributed privacy-preserving computation of medians. It is implemented by building at each iteration the (perturbed) histogram of the dimension being split and using the histogram in order to estimate the median. This histogram results from the privacy-preserving aggregation, on the platform, of the local histograms of workers. The following execution steps synthesize the computation of a perturbed histogram over the dimension d:

(1) Each worker locally instantiates his local histogram over the dimension d such that all the bins are set to 0 except the bin within which the worker's skill degree on d falls, set to 1.
(2) Each worker locally adds a *noise-share* to each bin, where a noise-share is a random variable such that the sum of a fixed number of noise-shares follows the two-sided geometric distribution. Recall (1) that the two-sided geometric distribution is infinitely divisible, and (2) that the addition to an integer of a random variable sampled from a two-sided geometric distribution well parameterized satisfies differential privacy (see above).
(3) Each worker encrypts each of its bins with the additively-homomorphic encryption scheme and sends it to the platform.

(4) The platform sums up all the encrypted histograms received from workers (bin per bin).
(5) The workers and the platform collaboratively decrypt the resulting encrypted histogram building on the threshold decryption feature of the encryption scheme (see above). The platform thus obtains the perturbed "summed up" histogram.
(6) The platform estimates the median based on the histogram, splits the dimension d around it, and either iterates on each of the two resulting partitions, or stops the algorithm if a termination criteria is met (e.g., sufficient number of splits).

The PKD algorithm further improves the quality of the counts of the hierarchy of partitions by post-processing them based on constrained inference techniques (see [Cormode et al. 2012; Hay et al. 2010] for details). A complete description of the PKD algorithm together with thorough experimental results are available in [Duguépéroux and Allard 2019].

2.3 Informed Task Tuning

The partitioning of the space of skills output by the PKD algorithm enables the computation of multi-dimensional COUNTs over the space of skills of the actual population of workers. A large variety of usages can be envisioned. We focus in this paper on a precise illustration that consists in publishing the partitioning of the space to requesters, and letting them tune their tasks according to the actual distribution of skills. For example, through an appropriate *task tuning helper*, provided by the platform or implemented on behalf of the requester, the latter could define wages according to the scarcity of a profile, or tune the skills required by a task such that they fit the profiles of a sufficiently high number of workers which results in lower pickup times.

3 DEMONSTRATION

This demonstration illustrates the PKD algorithm by (1) allowing its execution on a wide variety of parameters (e.g., various populations of workers, different numbers of iteration, different values of the ϵ privacy parameter) and (2) allowing the audience to create tasks matching the population of workers through a simple *task tuning* helper. The demonstration platform is centralized and simulates the distributed components of the PKD algorithm. We present below the technicals details of the demonstration platform, the parameters that can be set up by the audience (called *mutable* parameter) , the parameters that are fixed, and the demonstration scenario.

3.1 Platform

Figure 2 depicts the demonstration platform. The demonstration is implemented as a web application running through a single docker-compose file. In this file, several services handle every aspect of the application without any configuration or installation (except for docker and docker-compose, which are not specific to this demonstration). The first service and the core of the application is a Python Django web server used to serve the web interface and handle the commands issued by the demonstrator. The user interface, served by Django consists in simple HTML pages using the CSS framework Bootstrap for the design and ViewJS scripts for the dynamic components. A second service handles the long tasks in the background that cannot reasonably

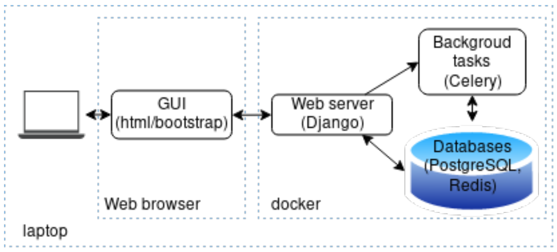


Figure 2: The demonstration runs on a single laptop executing the demonstration platform: the web server (Python Django), the background tasks handler (Celery) and the databases (PostgreSQL and Redis). The demonstration is accessible through a web interface (e.g., on the browser of the demonstration laptop).

be handled by Django without causing a loss of the user experience (i.e., tasks that last more than a few seconds such as the PKD algorithm and the CSV import of the workers). This service is written in Python with the Celery framework. Databases required by the application (PostgreSQL and Redis), to handle data persistency, are directly embedded in the docker-compose file. The homomorphic encryption features are disabled in order to reduce computation time for the demonstration. Indeed, for the sake of simplicity, all distributed operations, normally done by distinct workers, are done locally by the demonstration platform[7]. In particular, worker profiles are stored locally rather than being hold by individual workers, and the encrypted sum of histograms is replaced by a cleartext sum. These simplifications have no consequence on the output of the PKD algorithm. The source code of the demonstration is available publicly [8] and can be executed on a laptop where docker is installed (no configuration is required).

3.2 Parameters

In this demonstration we let the audience set various parameters, while others are fixed to default values. Default parameters are chosen to reflect plausible real-life settings while keeping the computation time reasonable. In particular, we limit the number of skills (e.g., 2 or 3 skills chosen by the audience) and the number of workers (e.g., a few hundreds instead of a few thousands in a real-life system).

The audience is able to set the skills, the worker profiles (either manually or automatically) and the PKD algorithm parameters. For these parameters, we also provide default values to help the audience: the differential privacy security parameter ($\epsilon = 0.1$), the number of splits for the partitioning ($splits = 7$) and the number of bins of histograms ($bins = 10$). Finally, the audience can tune tasks to fit with the previously defined workers.

Note that the termination criteria must be chosen carefully because it limits the number of splits of the space of skills[9]. The dimensions that are not part of the sequence will simply be ignored. The number of dimensions in workers profiles, and their respective priorities, is closely related to the application domain (e.g., How specific does the crowdsourcing process need to be?). In this paper, we make no assumption on the relative importance of dimensions.

[7]In a real-life scenario, encrypted operations would be performed in parallel by workers and the platform itself, which greatly reduces the costs
[8]https://gitlab.inria.fr/crowdguard-public/implems/pkd-demo
[9]It also impacts the overall computation time and quality of the estimation

3.3 Datasets

Three populations of workers are available by default: two populations are generated synthetically (i.e., through our `UNIF` worker generator that samples skill levels uniformly at random and our `ONESPE` worker generator that choses one strong skill uniformly at random for each worker and sets a low skill level to all others skills - see [Duguépéroux and Allard 2019] for details), and one population is computed from the public *Stackoverflow* dataset[10]. Additionally, the audience can instantiate a set of workers manually. Additional arbitrary populations of workers defined by the audience can be imported. Our platform accepts `CSV` files, such that each line is defined by three columns, as shown in Figure 3.

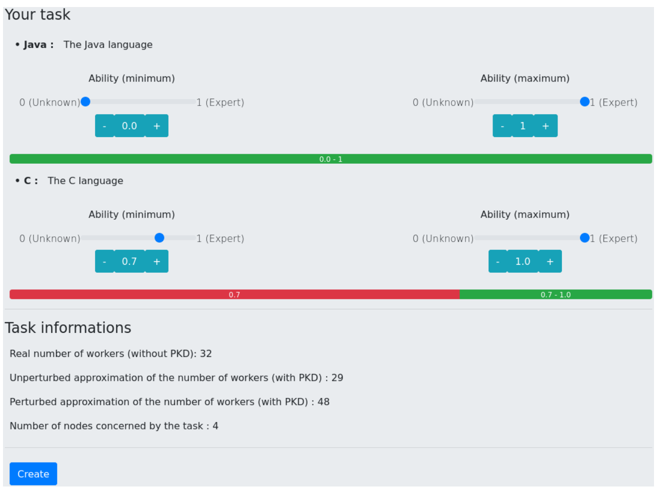
`UserID` (int)	`SkillID` (int)	`SkillLevel` (float in [0; 1])
12	3	0.4

Figure 3: Format of a worker dataset and illustration on a single worker.

3.4 Scenario

The demonstration scenario presents a simple execution sequence allowing the audience to observe the different steps of the PKD algorithm and to use the task tuning module. It concentrates on the task design, only the necessary information about the distribution of workers is displayed. A strong focus has been put on the simplicity and the clarity of the GUI which includes all the explanations needed to understand intuitively each step of the demonstration. The GUI is divided into a sequence of screens, where each screen is dedicated to a specific step of the execution sequence. First, an introduction screen presents the demonstration and its objective. Second, the audience choses the set of skills to consider. Third, the audience can launch the workers import (according to the various methods described above, including the import of a `CSV` file from the audience). Additional information about the distribution of skills within the dataset chosen is displayed through a Notebook document and commented. Fourth, the `PKD` algorithm is executed on the population of workers defined and outputs the space partitioning computed. Finally and most importantly, the audience uses our simple task tuning helper in order to tune a few tasks (1) without any information on the underlying population (default method within privacy-preserving crowdsourcing platforms) and (2) with the hierarchy of partitions computed by the `PKD` algorithm. Figure 4 shows the screen dedicated to tuning the task with information from the hierarchy of partitions. Optionnally, in order to observe the privacy/utility tradeoff, the audience is invited to explore the hierarchy of partitions and inspect the impact of the differentially private perturbation by comparing the perturbed counts to the exact unperturbed number of workers within each partition.

[10]We consider that a user is a worker, tags of posts are skills, and skill levels are a simple popularity score computed from the number of up-votes of each post. See https://gitlab.inria.fr/crowdguard-public/data/workers-stackoverflow for more details (i.e., description of the method and pre-processing scripts).



Task informations

Real number of workers (without PKD): 32

Unperturbed approximation of the number of workers (with PKD) : 29

Perturbed approximation of the number of workers (with PKD) : 48

Number of nodes concerned by the task : 4

Figure 4: Tuning the task with information from the hierarchy of partitions. On the first half of the screen (top), the skills requirements of a task are being tuned over the Java and `C` programming skills (the union of the leaf partitions appears on the green lines, and neighboring nodes appear on the red lines). On the second half of the screen (bottom), the screen displays information about the perturbed and actual number of workers corresponding to the task requirements.

ACKNOWLEDGMENTS

This work is partially supported by the French National Research Agency (Crowdguard project, ANR grant number `ANR-16-CE23-0004`).

REFERENCES

Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (1975), 509–517.

Louis Béziaud, Tristan Allard, and David Gross-Amblard. 2017. Lightweight privacy-preserving task assignment in skill-aware crowdsourcing. In *Proc. of DEXA '28.* 18–26.

Graham Cormode, Cecilia Procopiuc, Divesh Srivastava, Entong Shen, and Ting Yu. 2012. Differentially private spatial decompositions. In *Proc. of IEEE ICDE '12.* 20–31.

Ivan Damgård and Mads Jurik. 2001. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *International Workshop on PKC.* 119–136.

Joris Duguépéroux and Tristan Allard. 2019. *A Space Partitioning Algorithm for Privacy-Preserving Crowdsourcing.* Technical Report. (available on demand).

Cynthia Dwork. 2006. Differential privacy. In *Proc. of ICALP '06.* 1–12.

Arpita Ghosh, Tim Roughgarden, and Mukund Sundararajan. 2012. Universally utility-maximizing privacy mechanisms. *SIAM J. Comput.* 41, 6 (2012), 1673–1693.

Michael Hay, Vibhor Rastogi, Gerome Miklau, and Dan Suciu. 2010. Boosting the accuracy of differentially private histograms through consistency. *Proc. of the VLDB Endow.* 3, 1-2 (2010), 1021–1032.

Hiroshi Kajino. 2015. *Privacy-Preserving Crowdsourcing.* Ph.D. Dissertation. Univ. of Tokyo.

Matthew Lease, Jessica Hullman, Jeffrey P. Bigham, Michael S. Bernstein, Juho Kim, Walter Lasecki, Saeideh Bakhshi, Tanushree Mitra, and Robert C. Miller. 2013. Mechanical Turk is Not Anonymous. *SSRN Electronic Journal* (2013).

Ilya Mironov, Omkant Pandey, Omer Reingold, and Salil Vadhan. 2009. Computational Differential Privacy. In *Proc. of CRYPTO '29.* 126–142.

Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *Proc. of EUROCRYPT '99.* 223–238.

Huichuan Xia, Yang Wang, Yun Huang, and Anuj Shah. 2017. Our Privacy Needs to be Protected at All Costs: Crowd Workers' Privacy Experiences on Amazon Mechanical Turk. *Proc. of ACM HCI'17* 1 (2017), 113.