UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES 1
COMUE UNIVERSITÉ BRETAGNE LOIRE

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Informatique

Par

Joris DUGUÉPÉROUX

Protection des travailleurs dans les plateformes de crowdsourcing

Une perspective technique

Thèse présentée et soutenue à Rennes, le 01/10/2020
Unité de recherche : IRISA (UMR CNRS 6074), Université Rennes 1
Thèse N° :

Rapporteurs avant soutenance :

Caroline Fontaine	Directrice de recherche CNRS au LSV
Marc-Olivier Killijian	Professeur en informatique à l'UQÀM

Composition du Jury :

Président :	Alexandre Termier	Professeur en informatique, Université Rennes 1
Examinateurs :	Antonio Casilli	Professeur en sociologie à Télécom Paris
	Caroline Fontaine	Directrice de recherche CNRS au LSV
	Sébastien Gambs	Professeur en informatique à l'UQÀM
	Marc-Olivier Killijian	Professeur en informatique à l'UQÀM
	Daniel Le Métayer	Directeur de recherche Inria au Centre Inria Grenoble
Directeur de thèse :	David Gross-Amblard	Professeur en informatique, Université Rennes 1
Co-encadrant de thèse :	Tristan Allard	Maître de Conférences, Université Rennes 1

Acknowledgement

First and foremost, I'd like to thank academics. Of course, the *Université Rennes 1*, *Irisa*, and the *ENS Rennes*, that paid for my work and studies, but most importantly, all people I collaborated with. In this regard, I would like to thank the whole *DRUID* team for its support, and in particular PHD students from Rennes, both in *DRUID* and in other teams, with whom I enjoyed lots of discussions, meals and other worries about our future and publications. Victor Ibn Isis, Sujaya Maiyya, and all people in the *UCSB* with whom I enjoyed spending time with, and Amr El Abbadi and Mohammad Amiri, without whom I would not have gone so far on distributed issues for crowdsourcing. Sebastien Gambs, who accepted me as intern in 2016, and first taught me about differential privacy, and other technical aspects of ethics in databases and computer sciences. More generally, thanks to all of my PhD reviewers and jury (in alphabetical order of name), Antonio Casilli, Caroline Fontaine, Sébastien Gambs, Marc-Olivier Killijian, Daniel Le Métayer, for their patience reading the present manuscript and for their reviews. In particular, many thanks to Caroline Fontaine and Daniel Le Metayer, who followed me for all my PhD, and answered my worries at times I needed it. I'd also like to thank Antonin Voyez, who bravely accepted to improve my code as an intern (I hope I will never inflict such a pain upon him ever again), and my director, David Gross-Amblard, who helped a lot in so many little things I would not dare keeping the count of them. And of course, last but not least, my supervisor, Tristan Allard. I do not know how I could thank him enough for all the work he did, to help me improve myself, guide my research, and support me for these three years, all of that with so many constraints I would not imagine managing even half of it.

I also want to thank my teachers, whom I hold in the high regard. Thanks also to my students, who surprised me more often than I could count, by their jokes, by doing more than I expected, by their involvement, and always by their sympathy. Maybe even more importantly, thanks to all people that helped me build an opinion and a political view on my topic of study. To this regard, I would like to thank Antonio Casilli, Paola Tubaro, and the whole *DiPLab* team, for their studies. I'd also like to thank all invested people that contributed to make the issues of digital labor more visible, such as Lionel Maurel and

Laura Aufrère, or the teams behind *Cash Investigation*, *Le Média*, *Invisibles*, and I'm sure to forget a lot of them. Of course, thanks to all groups and unions supporting the rights of these workers. For opening my mind to many other related issues, whether on the computer science aspect, or on the social aspects, I also want to thank collaborators of the *RadX* seminars, whose contributions helped me a lot in both discovering and understanding stakes that I ignored, and widen my activities within the lab', in the best way possible. I would not be fair if I forgot the *FDLN* team, and its wonderful coordinators. Thank you, and thanks to all contributors, who made it possible, each year, to let people discover so many important topics, be them newcomers or advanced users. Thanks to the *Ingénieurs Engagés* and its members, for your resources, debates, disagreements, care, and other enjoyments together. Again, last but not least, thank you for the political philosophy workshop, which brought me so many things. Thanks in particular to Guillaume Fondu, Ali Kebir, Gabriel Mahéo, Mandie Joulin, Titus Masson, Even Perchais, Yangchen Bineau, and so many others.

Finally, I absolutely want to thank all of my friends and family members. In Cachan, thanks to all *A*0 (adoption included), to the procrastination heaven, and to all *A*♡, who will certainly recognize themselves. Thanks to Estelle Legros, for our numerous, always interesting, and never completely abandoned discussions. In Rennes, thanks to all my friends with whom I played a lot (some would say too much, some not enough), and especially to Jules Combot, who helped me rediscover the long forgotten pleasure of role-playing games and deep late-night discussions, and to Mandie Joulin, who shares my everyday life with games, cats, Veblen, love, knitting, food, and other national accounts. In Poitiers, thanks to all my friends. I would not dare to cite them all as I certainly would forget some, but I'm confident that they will recognize themselves. Thanks in particular to François Courset, for everything we shared and still share, and to Émilie Soulard, who even came to Rennes for a whole year, and never missed a chance to ask for news. And of course, thanks to my family, both close and further. Thanks to my mother and her everlasting worries for my well-being and help on so many administrative issues, to my father for his advice and help, and to my sister, who always supported me and never hold it against me for all these years.

Thanks to all of you, and all those I forgot. Without you, all this work would not have been possible.

Table of Contents

Résumé en français **9**

Publications during the PhD **16**

1 Introduction **19**
1.1 Did you say "crowdsourcing"? . . . 20
1.1.1 What is crowdsourcing? . . . 20
1.1.2 Why do we use the expression "crowdsourcing"? . . . 21
1.1.3 Crowdsourcing in Computer Science . . . 23
1.2 Real-life issues with crowdsourcing . . . 24
1.2.1 Crowdsourcing in the news and in the law . . . 24
1.2.2 Crowdsourcing in Social Sciences . . . 25
1.3 Our answers . . . 27

2 From task tuning to task assignment in privacy-preserving crowdsourcing platforms **31**
2.1 Introduction . . . 31
2.2 Related work . . . 36
2.2.1 Privacy-preserving space partitioning . . . 36
2.2.2 Privacy-preserving task assignment . . . 39
2.2.3 Task design . . . 42
2.3 Preliminaries . . . 43
2.3.1 Participants model . . . 43
2.3.2 Privacy tools . . . 44
2.3.3 Quality measure . . . 52
2.4 The `PKD` algorithm . . . 52
2.4.1 Computing private medians . . . 53
2.4.2 Global execution sequence . . . 56
2.4.3 Complexity analysis . . . 59
2.4.4 Security analysis . . . 60

2.5 Privacy-preserving task assignement . . . 61
2.5.1 PIR for crowdsourcing: challenges and naive approaches . . . 62
2.5.2 PIR partitioned packing . . . 63
2.5.3 Optimizing the packing . . . 67
2.6 Experimental validation . . . 71
2.6.1 Datasets . . . 71
2.6.2 PKD algorithm . . . 73
2.6.3 Assignment using packing . . . 76
2.7 Discussion . . . 81
2.7.1 Updating tasks and PIR libraries . . . 81
2.7.2 Updating PKD . . . 83
2.8 Demonstration . . . 84
2.8.1 Platform . . . 84
2.8.2 Parameters . . . 85
2.8.3 Datasets . . . 86
2.8.4 Scenario . . . 86
2.9 Conclusion . . . 88

3 SEPAR: a privacy-preserving approach to regulate crowdsourcing 89
3.1 Introduction . . . 89
3.2 Related work . . . 92
3.2.1 Decentralized systems . . . 92
3.2.2 Global crowdsourcing processes . . . 94
3.3 Problem formulation . . . 95
3.3.1 Motivating example . . . 95
3.3.2 Crowdsourcing environment . . . 96
3.3.3 Security model . . . 98
3.3.4 Problem . . . 100
3.4 Expressing global regulations . . . 101
3.4.1 Expressing constraints . . . 101
3.4.2 Expressing requests for certificates . . . 102
3.5 Enforcing global regulations . . . 103
3.5.1 Implementing a token-based system . . . 103
3.5.2 Task processing sequence . . . 107

3.5.3 Privacy analysis . . . 108
3.6 Coping with distribution . . . 112
3.6.1 Blockchain ledger . . . 113
3.6.2 Consensus in SEPAR . . . 115
3.7 Experimental evaluations . . . 123
3.7.1 Token generation . . . 124
3.7.2 Impact of cross-platform tasks . . . 124
3.7.3 Varying the types of constraints . . . 126
3.7.4 Varying the number of platforms . . . 127
3.8 Conclusion . . . 128

4 Towards extending PIR to multiple downloads 129
4.1 Introduction . . . 129
4.2 Related Work . . . 132
4.2.1 Computational PIR . . . 132
4.2.2 Information theoretical PIR . . . 133
4.3 Attacks . . . 134
4.3.1 Model . . . 134
4.3.2 Attack . . . 138
4.4 Counter-measures . . . 138
4.4.1 Packing . . . 140
4.4.2 Mitigating the attack . . . 141
4.5 Future work . . . 145
4.6 Conclusion . . . 146

5 Conclusion and future work 147
5.1 Summary of contributions . . . 147
5.2 Future work . . . 148

Bibliography 151

Résumé en français

Chapitre 1 : Introduction

Ce doctorat s'inscrit dans le cadre du projet ANR Crowdguard [1], qui a pour but de protéger les participants aux plateformes de crowdsourcing. Dans cette thèse, nous entendons par *crowdsourcing* l'ensemble des interactions qui régissent trois types d'acteurs que sont les travailleurs, les employeurs et les plateformes, dans le cadre d'une relation de travail : les employeurs ont des tâches à effectuer, et passent par les plateformes en ligne pour rémunérer les travailleurs qui vont les effectuer. La protection, quant à elle, a de nombreuses facettes : de nombreux abus sont référencés à la fois dans la presse, dans les jurisprudences, et dans les travaux de recherche en sociologie, et nous nous focalisons dans ce travail sur certains d'entre eux.

Parmi les problèmes fréquemment cités, deux se distinguent particulièrement : d'une part la surveillance de masse effectuée par les plateformes sur les travailleurs, et d'autre part le statut ambigu des travailleurs, souvent qualifiés non pas d'employés mais d'indépendants, ce qui pose de nombreuses questions vis à vis du manque de protection dont bénéficie ce dernier statut. Dans cette thèse, nous nous focalisons en particulier sur la protection de la vie privée des travailleurs dans le cadre de leurs interactions avec la plateforme, et sur leur protection légale, par la mise en place d'outils permettant de réguler les plateformes et de fournir des preuves d'éventuels abus. Les domaines abordés étant assez variés, et par souci de lisibilité, nous proposons dans chaque chapitre l'état de l'art correspondant. Le Chapitre 2 est consacré à la création de versions respectueuses de la vie privée d'algorithmes utiles aux plateformes. Nous explorons au Chapitre 3 la création d'outils de régulation (également respectueux de la vie privée) afin de faciliter l'application des décisions des législateurs. Nous nous intéressons dans le Chapitre 4 aux limites du PIR (*Private Information Retrieval*, dont nous n'avons pas trouvé de traduction appropriée en français), technique utilisée dans le Chapitre 2, bien que celles-ci soient relativement indépendantes des problématiques liées au crowdsourcing. En effet, bien que cette optique ne semble pas être étudiée dans l'état de l'art sur le sujet, une utilisation naïve du PIR

1. `https://crowdguard.irisa.f/`

à de multiples reprises engendre des problèmes de sécurité qui ne sont pas présents lors d'une utilisation unique. Enfin, le Chapitre 5 conclut notre manuscrit, en résumant les contributions et en proposant des pistes pour des travaux futurs.

Chapitre 2 : Protéger la vie privée lors de l'affectation de tâches et de calculs statistiques sur les données personnelles

Dans ce chapitre, nous nous intéressons à deux manières d'utiliser les données personnelles des travailleurs, ici comprises comme leurs niveaux de compétence dans différents domaines susceptibles d'être utiles dans le crowdsourcing. Ces données sont fréquemment utilisées en crowdsourcing, que ce soit pour effectuer une affectation des tâches aux travailleurs (utilisation que nous qualifions de *primaire*) ou pour calculer des statistiques, utilisables à diverses fins (utilisations *secondaires*, par exemple, pour que la plateforme puisse faire la promotion de la population de travailleurs qu'elle a à sa disposition, ou pour que les employeurs puissent concevoir leurs tâches au mieux en fonction de cette population). Le travail exposé dans ce chapitre permet d'éviter la surveillance des travailleurs, tout en autorisant ces utilisations légitimes, ce que ne permet pas de faire l'état de l'art en toute généralité, du fait de limites dans le contexte étudié (crowdsourcing basé uniquement sur la géolocalisation), de temps de calcul irréalistes pour des applications concrètes, ou une qualité trop faible.

Tout d'abord, nous nous intéressons aux usages secondaires. Pour pouvoir fournir les statistiques nécessaires à ces usages tout en protégeant la vie privée des travailleurs, nous avons conçu l'algorithme `PKD` (pour *Private KD-tree*). Cet algorithme garantit, en terme de sécurité, que les données rendues accessibles à un attaquant dit *honnête mais curieux* sont à tout instant au minimum soit chiffrées soit protégées par la confidentialité différentielle (*differential privacy*), en utilisant pour cela de l'ajout de bruit aléatoire pour la confidentialité différentielle, ainsi que des méthodes venant de la cryptographie. La démarche globale pour les usages secondaires est résumée dans la Figure 1.

Pour résumer, nous calculons un partitionnement de l'espace des compétences des travailleurs, en fonction de la distribution de ceux-ci. La méthode générale de partitionnement choisie, le *KD-tree*, coupe successivement chaque dimension au niveau de la médiane des points représentés (ici, la représentation graphique des profils des travailleurs). Pour

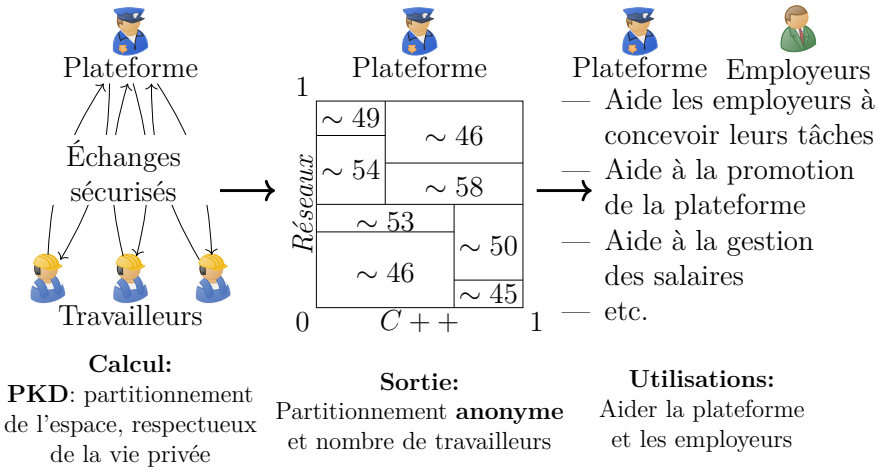


Figure 1 – Illustration de l'algorithme `PKD`: autoriser des usages secondaires de profils des travailleurs en protégeant leur vie privée

effectuer ces coupes sans connaître la position des points (pour protéger les données), nous calculons des histogrammes de manière distribuée, en utilisant pour cela du chiffrement homomorphe qui, couplé à la confidentialité différentielle, permet de garantir que les résultats dévoilés seront protégés. Au final, le résultat obtenu est une distribution bruitée (bien qu'elles restent statistiquement correctes, les valeurs affichées peuvent différer de la réalité afin de satisfaire les contraintes de confidentialité différentielle) mais respectueuse de la vie privée des travailleurs dans l'espace des compétences. Cette approche est illustrée par une démonstration [2], qui implémente l'approche sur un scénario dans lequel des employeurs cherchent à concevoir des tâches adaptées à la population de travailleurs. Elle est aussi validée par des preuves de sécurité (confirmant que cette approche est sécurisée contre des attaquants dits "honnêtes mais curieux"), ainsi que des expérimentations sur la qualité des distributions obtenues et sur les temps de calcul, permettant de conclure que la qualité de la distribution obtenue est comparable aux résultats de l'état de l'art en contexte centralisé, et que le temps de calcul est viable pour des scénarios réalistes.

Ensuite, nous nous intéressons à l'utilisation dite *primaire*, qui est une utilisation plus directement en lien avec le crowdsourcing : l'affectation des travailleurs aux tâches. Pour ce faire, nous utilisons essentiellement le PIR, qui permet à un utilisateur de télécharger des données sans que l'hébergeur ne sache quelles données ont été téléchargées. Cette technique est cependant limitée car ses garanties ne s'appliquent plus lors de multiples téléchargements successifs, ce qui la rend inapplicable en crowdsourcing pour un travailleur

2. disponible publiquement ici : `https://gitlab.inria.fr/crowdguard-public/implems/pkd-demo`

souhaitant télécharger plusieurs tâches. Pour cette raison, des adaptations de ces protocoles, tenant compte de nos contraintes, ont été conçues. L'idée principale est de télécharger un sur-ensemble des tâches réellement souhaitées en une seule fois, afin qu'il soit impossible de déduire quoi que ce soit du nombre de téléchargements. Le KD-tree précédemment construit est alors utilisé pour limiter le plus possible les téléchargements inutiles. Cette approche est elle aussi validée par des preuves de sécurité et des expérimentations portant sur le temps de calcul et la qualité des affectations, permettant là encore de conclure que les temps de calcul sont raisonnables à grande échelle.

Dans l'ensemble, ce chapitre a mené à de multiples publications au long de la thèse [3] [4] [5] [6] [7] [8] [9].

Chapitre 3 : Réguler le crowdsourcing tout en protégeant la vie privée avec SEPAR

Le manque de régulation dans le milieu du crowdsourcing est fréquemment dénoncé par les travailleurs, syndicats, médias, ou travaux en sociologie, et la loi reste parfois inappliquée. Cependant, il est notable qu'il n'existe à ce jour aucun outil permettant d'appliquer une régulation uniformisée sur l'ensemble des plateformes tout en protégeant la vie privée des participants. Dans ce Chapitre 3, nous proposons donc un outil, SEPAR, qui permet de répondre à cette problématique. Nous considérons une situation dans laquelle de multiples plateformes, travailleurs et employeurs sont présents, et interagissent : un travailleur peut être présent sur plusieurs plateformes, et faire appel à plusieurs employeurs,

3. Joris Duguépéroux and Tristan Allard, « From Task Tuning to Task Assignment in Privacy-Preserving Crowdsourcing Platforms », *in*: *Transactions on Large-Scale Data and Knowledge-Centered Systems* (2020), (40 pages).

4. Joris Duguépéroux and Tristan Allard, « Privacy-Preserving Informed Task Design in Crowdsourcing Processes », *in*: *BDA'19*, (10 pages), 2019.

5. Joris Duguépéroux, Antonin Voyez, and Tristan Allard, « Task-Tuning in Privacy-Preserving Crowdsourcing Platforms », *in*: *Proc. of EDBT'20*, (4 pages), 2020, pp. 623–626.

6. Joris Duguépéroux, « Guaranteed Confidentiality and Efficiency in Crowdsourcing Platforms », *in*: *APVP'19*, (2 pages), 2019.

7. Joris Duguépéroux, « Guaranteed Confidentiality and Efficiency in Crowdsourcing Platforms », *in*: *BDA'17*, (2 pages), 2017.

8. Joris Duguépéroux and Tristan Allard, « Un algorithme de partitionnement de l'espace des compétences pour plateformes de crowdsourcing respectueuses de la vie privée », *in*: *HIA'20*, (17 pages), 2020.

9. Joris Duguépéroux, Antonin Voyez, and Tristan Allard, « Task-Tuning in Privacy-Preserving Crowdsourcing Platforms », *in*: *HIA'20*, (8 pages), 2020.

et inversement pour les employeurs. Bien que des travaux traitant de problèmes analogues aient pu être présentés dans des contextes bancaires ou avec des chaînes de blocs protégeant les informations personnelles, ceux-ci ne permettent pas de résoudre notre problème, qui mélange trois types distincts de participants ainsi que des contraintes de régulation, transparence et confidentialité.

Dans ce contexte, nous proposons un système distribué permettant d'appliquer des règles génériques et simples à exprimer, qui autorise la régulation des échanges entre les participants en garantissant à la fois transparence et respect de la vie privée. Ces règles peuvent prendre la forme de contraintes, comme interdire aux travailleurs de travailler plus de 40h par semaine sur l'ensemble des plateformes, ou limiter le nombre de tâches effectuées sur une certaine plateforme pour favoriser la concurrence. Ces contraintes sont alors exprimées par un nombre arbitraire de règles spécifiant (ou non) des participants (travailleurs, employeurs ou plateformes) et interdisant à ces participants toute interaction au delà d'un certain seuil. Une autre forme de règle possible est la demande de certification, par exemple pour qu'un travailleur puisse prouver le nombre de tâches effectuées afin de cotiser à des droits sociaux, ou dans le cadre d'une poursuite en justice d'une plateforme pour une requalification en emploi salarié. Cette demande de certification spécifie des participants (au moins un) et un seuil, et garanti que les participants spécifiés ont interagi un nombre de fois supérieur au seuil indiqué.

Pour obtenir ce résultat, nous proposons un système distribué dont les éléments clés sont les suivants :

— Les plateformes constituent des nœuds d'interactions : ils peuvent être constitués de multiples serveurs, et les décisions et actions de la plateforme résulteront du consensus dit "local" de ces serveurs
— Un registre global, accessible publiquement et concrétisé sous la forme d'une chaîne de blocs à permissions (*permissioned blockchain*), est mis à jour à chaque étape par consensus global entre l'ensemble des plateformes afin d'enregistrer des traces des transactions, permettant de garantir la transparence sans compromettre la vie privée
— Des consensus inter-plateformes, lorsque des tâches concernent de multiples plateformes (par exemple, un employeur pourrait vouloir soumettre une tâche à de multiples plateformes concurrentes)

Ces trois formes de consensus régissent l'utilisation de jetons (*token*) qui sont garants des règles décidées. Ces jetons sont émis initialement par une entité de confiance, telle

qu'une autorité administrative par exemple, qui est en charge de garantir l'unicité de tous les participants (éviter la création de multiples comptes), de distribuer les jetons, et de trancher les désaccords en cas de contestation. Deux types de jetons existent. D'abord, les jetons de contraintes, distribués en nombre contrôlé, et utilisés à chaque tâche, qui permettent de s'assurer qu'aucun participant n'excède le quota d'interactions autorisées (les jetons ne contiennent pas d'informations liées à l'identité, mais sont diffusés publiquement sur le registre global et leur authenticité peut être vérifiée, de sorte qu'il est impossible qu'un participant agisse une fois tous ses jetons utilisés). Ensuite, les jetons de certificat, eux aussi utilisés à chaque tâche, comportant une partie publique diffusée sur le registre global, et une partie privée, liée à son identité ainsi qu'à celle des autres participants à une même tâche, et qui permet de prouver son implication dans une tâche (une fois la partie publique du jeton validée).

De cette manière, chaque participant peut certifier son implication dans les tâches avec lesquelles il est lié (ainsi que la présence d'autres participants), tout en étant limité dans le nombre des tâches par des contraintes décidées par une autorité administrative.

En terme de sécurité, nous considérons ici un attaquant caché, c'est à dire qu'il peut agir malicieusement si cela n'est pas détecté, mais qu'il s'abstiendra s'il sait qu'il sera détecté. Avec ce modèle d'attaque, nous exigeons (pour simplifier) qu'aucun participant, à l'exception de l'autorité administrative, ne puisse apprendre quoi que ce soit d'une interaction avec laquelle il n'a aucun lien. Enfin, des expérimentations sur le temps de calcul et le passage à l'échelle, ainsi que des preuves de sécurité permettent de voir que le temps de calcul augmente avec le nombre d'interactions (notamment lorsque les tâches sont nombreuses et toutes sur plusieurs plateformes), tout en restant applicable à des scénarios réalistes.

Ce travail sera bientôt soumis pour une publication à la conférence SIGMOD [10].

Chapitre 4 : Extension du PIR aux téléchargements multiples

Dans le Chapitre 4, nous nous intéressons plus en profondeur au problème du PIR déjà abordé dans le premier chapitre : considérant une liste de contenus, cette technique permet

10. Mohammad Javad Amiri, Joris Duguépéroux, et al., *SEPAR: A Privacy-Preserving Blockchain-based System for Regulating Multi-Platform Crowdworking Environments*, Soon submitted to the SIGMOD conference (14 pages), 2020, URL: `https://arxiv.org/abs/2005.01038`.

à un utilisateur de télécharger l'un de ces contenus sans que l'hébergeur ne sache lequel a été choisi. Cependant, les garanties de cette technique ne s'étendent pas au-delà d'un unique téléchargement, ce qui limite fortement son usage pour des applications concrètes. Pourtant, nous n'avons pas pu trouver dans la littérature actuelle de mentions claires de cette limite, qu'il s'agisse de l'attaquer ou de la défendre. Ce chapitre, moins formel et approfondi que les précédents, propose tout d'abord une attaque possible, en se basant sur des connaissances extérieures sur les contenus ainsi que sur le nombre de téléchargements effectués. Dans un second temps, nous proposons des pistes de défense afin de pallier ce problème et d'autoriser le PIR pour effectuer de multiples téléchargements sans perdre en sécurité. Malgré le manque de preuves et de validations expérimentales, ce travail ouvre des pistes intéressantes sur de nouveaux modèles de sécurité et offre un aperçu des différentes solutions susceptibles d'être développées à l'avenir.

Chapitre 5 : Conclusion et travaux futurs

Dans ce dernier chapitre, nous synthétisons nos apports et contributions, et proposons des pistes pour les prolonger. Ces pistes peuvent s'étendre dans deux directions principales. D'une part, une continuation directe des objectifs de nos travaux est possible, par exemple pour améliorer la qualité des affectations et statistiques dans le Chapitre 2, pour augmenter l'expressivité des contraintes du Chapitre 3, ou pour valider théoriquement ou expérimentalement les hypothèses émises dans le Chapitre 4. Cependant, une autre continuation est possible, qui consisterait à étendre encore le champ des protections apportées aux travailleurs. Par exemple, il n'y a que très peu de travaux dans la littérature traitant de l'équité entre les travailleurs, de la transparence des traitements individuels, ou de la portabilité des profils et notes reçues d'une plateforme à une autre. Ces points en suspens sont pourtant autant de problématiques qui concernent directement les travailleurs dans le crowdsourcing, et nous semblent donc importantes à souligner.

List of Publications

Paper soon under review

— Mohammad Javad Amiri, Joris Duguépéroux, et al., *SEPAR: A Privacy-Preserving Blockchain-based System for Regulating Multi-Platform Crowdworking Environments*, Soon submitted to the SIGMOD conference (14 pages), 2020, URL: `https://arxiv.org/abs/2005.01038`

Peer-reviewed international journal article

— Joris Duguépéroux and Tristan Allard, « From Task Tuning to Task Assignment in Privacy-Preserving Crowdsourcing Platforms », *in*: *Transactions on Large-Scale Data and Knowledge-Centered Systems* (2020), (40 pages)

Peer-reviewed national conference article

— Joris Duguépéroux and Tristan Allard, « Privacy-Preserving Informed Task Design in Crowdsourcing Processes », *in*: *BDA'19*, (10 pages), 2019

Peer-reviewed demonstrations, PhD papers and workshops

— Joris Duguépéroux, Antonin Voyez, and Tristan Allard, « Task-Tuning in Privacy-Preserving Crowdsourcing Platforms », *in*: *Proc. of EDBT'20*, (4 pages), 2020, pp. 623–626

— Joris Duguépéroux, « Guaranteed Confidentiality and Efficiency in Crowdsourcing Platforms », *in*: *APVP'19*, (2 pages), 2019

— Joris Duguépéroux, « Guaranteed Confidentiality and Efficiency in Crowdsourcing Platforms », *in*: *BDA'17*, (2 pages), 2017

— Tristan Allard, Tassadit Bouadi, et al., « From Self-Data to Self-Preferences: Towards Preference Elicitation in Personal Information Management Systems », *in*: *Proc. of PAP'17*, (6 pages), 2017

— Joris Duguépéroux and Tristan Allard, « Un algorithme de partitionnement de l'espace des compétences pour plateformes de crowdsourcing respectueuses de la vie privée », *in*: *HIA'20*, (17 pages), 2020

— Joris Duguépéroux, Antonin Voyez, and Tristan Allard, « Task-Tuning in Privacy-Preserving Crowdsourcing Platforms », *in*: *HIA'20*, (8 pages), 2020

Chapter 1

Introduction

Originally, this PhD was described as a "research project dedicated to designing models and algorithms for the enforcement of strong protection measures of the individuals participating in crowdsourcing processes dedicated to work". In this description [1], two main concepts are combined: crowdsourcing, and protection, both kept relatively undefined. Crowdsourcing, as a name, is a contraction of "crowd" and "outsourcing", and was first introuced by Howe [2], in 2006. This word is used in many senses, and in this work, we define it as follows: crowdsourcing refers to the ecosystem which encompasses professional and economic interactions between three parties: *requesters*, who have tasks to accomplish, *workers*, who are willing to do these tasks in exchange for a reward and constitute the "crowd", and *platforms*, which are intermediaries. With this definition are included well-known companies, such as Uber or Amazon Mechanical Turk. The protection part is no easier to define: who is to be protected, and from what? This work mostly focuses on the protection of workers. More specifically, this manuscript focuses on providing sound protection of the privacy of workers (without compromising efficiency of usual crowdsourcing algorithms) and access to legal protection (*e.g.* being able to defend their cases in a court, or benefiting from legal restrictions).

To introduce this work, we first try to understand with more depth the concept of crowdsourcing in Section 1.1: what are the assumptions of this concept, why do so many concurrent names exist, and how is it studied in Computer Science? The reality behind crowdsourcing is then explored in Section 1.2: what kind of issues can be seen in the news, what legislations apply in this economic model, and what Social Sciences can say about. Finally, we introduce our contributions in Section 1.3, and explain how they articulate to answer the issues that have been found.

1. From the Crowdguard ANR Project, which funded the PhD `https://crowdguard.irisa.fr/`
2. Jeff Howe, « The rise of crowdsourcing », *in*: *Wired magazine* 14.*6* (2006), pp. 1–4.

1.1 Did you say "crowdsourcing"?

As stated previously, crowdsourcing refers to the idea of outsourcing work to a potentially unknown crowd. However, this succinct definition hides many realities. We propose here to explore first the objectives of crowdsourcing, the way it is treated in Computer Science, and a few of the numerous words that refer to this phenomenon, which reflect different points of view and nuances.

1.1.1 What is crowdsourcing?

While outsourcing assumes that it is more efficient to have work done by people in other countries or by intermediate companies, crowdsourcing assumes that some tasks exist, that are more efficiently done by potentially numerous unknown people than by a limited number of known employees. The metrics used to measure efficiency, in this context, is often profitability. For instance, in outsourcing, profitability may be increased by low-paid workers and less protective legislations. Although legislations play an important role for crowdsourcing too, as we will see later, this is not the only element which makes it more profitable: the diversity of the crowd (real or assumed) also plays an important role. Diversity can be interesting for tasks that have specific requirements to be accomplished (*e.g.* close geolocation to optimize transportation, understanding a given language for translations, etc.), for market studies on specific crowds or even for art contests. All these tasks require workers, but hardly justify recruitment.

To fully understand what crowdsourcing means, it is interesting to distinguish its use-cases. Many typologies have been proposed to distinguish various kinds of crowdsourcing processes in the last decade[3][4][5][6]. In particular, the typology used by Casilli in 2019[7] proposes the three following categories:

— *On-demand services*, which propose services to individual requesters, such as transportation or delivery. We also include freelancing in this category.

— *Micro-tasks*, which refer to tasks that do not require advanced skills, are numerous,

3. Thierry Burger-Helmchen and Julien Pénin, « Crowdsourcing: définition, enjeux, typologie », *in*: *Management & Avenir 1* (2011), pp. 254–269.

4. Anhai Doan, Raghu Ramakrishnan, and Alon Y Halevy, « Crowdsourcing systems on the world-wide web », *in*: *Communications of the ACM* 54.*4* (2011), pp. 86–96.

5. David Geiger, Michael Rosemann, et al., « Crowdsourcing Information Systems - Definition, Typology, and Design », *in*: *Proc. of ICIS'12*, 2012.

6. Antonio A Casilli, *En attendant les robots-Enquête sur le travail du clic*, Le Seuil, 2019.

7. *Ibid.*

quick to perform and poorly paid, such as building datasets to train artificial intelligences (*e.g.* recognizing patterns in pictures, transcribing discussions, etc.), which is easy for humans, but harder for algorithms.

— *Social media labor*, which includes all actions performed on a social media (*e.g.* liking a content, adding content or posts, visiting any webpage, or anything that can be measured on these social media), seen as work in the way that they are used to produce value by the owner of the network, although "workers" are not paid in the process.

With our previous definition, we voluntarily exclude social media labor from our scope, as requesters and platforms, in this context, are hard to distinguish, when they are distinct at all. It is interesting to notice that the nature of task is not the only difference between on-demand services and micro-task. Indeed, even the assignment process varies a lot between these two categories. For micro-tasks, the identity of the worker does not really matter for the requester, as the task does not require specific skill, and no real-life contact is made. Therefore, workers are made as invisible as possible by the platform: the task is often published on the platform and chosen by workers [8]. On the opposite, for on-demand services, requesters often have the last word to decide whether or not they accept a worker, because of the required skills, or to ease the client for real-life contacts: for freelancing, the choice is often let to the requesters to decide which worker they hire, according to their presentation or curriculum vitae [9].

1.1.2 Why do we use the expression "crowdsourcing"?

Although "crowdsourcing" is relatively widespread in Computer Science, it is far from being the only word that refers to this phenomenon. We explore here a few of these names, along with their connotations and variations.

In many cases, the meaning of the word is straightforward: for instance, "playbour" insists on the fact that online work may look like a game on platforms, and "crowdsourcing" insists on the fact that work can be moved to an unknown crowd. Similarly, the "gig economy" [10] insists on variations on the economy and the way it works, making abstraction of the life of workers and of the tasks, to focus on the fact that these tasks are smaller and

8. See for instance platforms such as Amazon Mechanical Turk, Appen or Kicklox
9. Examples include fiverr, 99designs, or Upwork
10. Valerio De Stefano, « The rise of the just-in-time workforce: On-demand work, crowdwork, and labor protection in the gig-economy », *in*: *Comp. Lab. L. & Pol'y J.* 37 (2015), p. 471.

smaller, that hardly require full-time employees, so that workers have to adapt and to be more "agile". "Platform capitalism" [11] is also relatively clear in its meaning, describing the rise of powerful platforms, with increased power on job market.

However, meanings and nuances are not always that obvious, and it is also interesting to focus on the word used to refer to the work itself. Indeed, as reminded from Marxian philosophy of work, work is a many-faceted notion [12] [13], as it refers to (1) the relationship between the worker and the task, the qualitative content of a productive activity, (2) the relationship between the worker and others, or in other words, the role of human activity in production, measured quantitatively and independently from its concrete properties (*e.g.* through money or the work time), and (3) the professional identity which determines social status. In English, "work" is used to express the first connotation, "labor" the second, and "job" the last one.

Although it is incomplete, this grid of analysis provides tools to understand some possible implicit nuances. "Freelancing job" uses *job* as an independent unit of work, but can also be seen as a social status given to workers: platforms give them a job that they do not have to be ashamed of. "Digital labor", used in sociology, focuses on the living conditions of these workers, insisting on the fact that these activities do create value, although this value is appropriated by platforms. "Crowdwork", simply states that tasks are performed by the crowd. "Ghost work" [14], focuses on the fact that tasks, and not only workers, are hidden from the public sight. Interestingly, "future of work" [15] [16], can also be analyzed through this prism. This expression focuses on the evolution of tasks, how to perform them differently, and what their future will be. However, this syntactic analysis of the "future of work" does not say anything about workers. Although the authors using this expression all have an opinion on what the future of workers should be (and not only work), this opinion is hard to deduce only from the expression itself (*e.g.* whether this future is meant to improve their lives or to focus on profitability): not all expressions were created using the Marxian distinction between work, labor and jobs in mind, and the corresponding analysis is not relevant in all contexts.

11. Nick Srnicek, *Platform capitalism*, John Wiley & Sons, 2017.
12. Casilli, *op. cit.*
13. Dominique Méda, *Le Travail. une valeur en voie de disparition?*, Flammarion, 2010.
14. Mary L Gray and Siddharth Suri, *Ghost Work: How to Stop Silicon Valley from Building a New Global Underclass*, Eamon Dolan Books, 2019.
15. FoW participants, *Imagine all the People and AI in the Future of Work*, ACM SIGMOD blog post, 2019.
16. Janine Berg, Marianne Furrer, et al., *Digital labour platforms and the future of work : Towards decent work in the online world*, tech. rep., International Labour Organization, 2018.

In this manuscript, we mostly use the word "crowdsourcing". This choice does not reflect our opinion or focus on the phenomenon, but a more pragmatical choice: to the best of our knowledge, "crowdsourcing" is the most commonly used wording in Computer Science to refer to this phenomenon, therefore, it appears to be both easier and more practical to use it for communications.

1.1.3 Crowdsourcing in Computer Science

In recent years, works focusing on crowdsourcing have increased a lot in Computer Science. We propose here a very succinct overview of various ways in which Computer Science studies crowdsourcing.

According to a survey provided by Chittilappilly et al. [17], it is clear that an important focus is given on helping requesters and platforms in crowdsourcing. In particular, this survey highlights the diversity of techniques used in crowdsourcing, in incentives for workers, decompositions of tasks, assignment, or quality control. A survey from Li et al. [18] makes this focus even clearer: their analysis on crowdsourcing efficiency determines that most of the problems come from the workers. Indeed, the three problems they identify in crowdsourcing are "(1) Quality Control: Workers may return noisy or incorrect results [...] (2) Cost Control: The crowd is not free [...] (3) Latency Control: The human workers can be slow [...]".

However, many computer scientists are also interested in other aspects of crowdsourcing. For instance, a survey from Amer-Yahia et al. [19] focuses on what its authors call *worker-centric* aspects of crowdsourcing, saying that "it is essential to shift from requester-centric optimizations to an approach that integrates what workers want from a crowdsourcing platform". This survey studies among other things the design of incentives or feedbacks, various ways to assign tasks to workers (for both "micro-tasks" and "collaborative tasks"), and, more generally, how to optimize crowdsourcing processes by taking workers' psychology into account.

Finally, some works focus on privacy and protection of crowdsourcing in general. Some

17. Anand Inasu Chittilappilly, Lei Chen, and Sihem Amer-Yahia, « A Survey of General-Purpose Crowdsourcing Techniques », *in*: *IEEE Transactions on Knowledge and Data Engineering* 28.*9* (2016), pp. 2246–2266.

18. Guoliang Li, Jiannan Wang, et al., « Crowdsourced data management: A survey », *in*: *IEEE TKDE* 28.*9* (2016), pp. 2296–2319.

19. Sihem Amer-Yahia and Senjuti Roy, « Toward worker-centric crowdsourcing », *in*: (2016).

of them focus on the protection of tasks, seen as sensitive [20] [21], while other rather focus on the protection of workers, so that they can work without risks [22] [23] [24] [25]. These works are not interested in improving profit of workers, platforms or requesters: they rather see crowdsourcing as a potential source of danger, and try to enforce protections through a wide diversity of techniques. In this manuscript, our work follows this last approach.

1.2 Real-life issues with crowdsourcing

In this section, we focus on the issues that have been seen in crowdsourcing in the last decade. These limits, both legal and ethical, are frequently mentioned in the news and in the law, but can also be analyzed more precisely thanks to Social Sciences.

1.2.1 Crowdsourcing in the news and in the law

In recent years, crowdsourcing has drawn more and more attention. In France, the importance of crowdsourcing has risen so much that even the President gave his opinion in 2016 on the situation of Uber drivers and their legal status in an interview [26]. This discussion took place with a Uber trade-unionist, worried about the living conditions of workers, and their legal situation as independent contractors, and not employees from Uber. Indeed, using the status of independent contractors allows platforms to avoid regulations that exist on workers or employees, often heavier and more protective. As summed up by Toxtli et al. [27], "the question of whether a worker is an employee or an independent contractor is important because a worker who is an employee is entitled to many rights and protections, including minimum wage, overtime pay, paid vacation, sick pay, employer-subsidized health insurance, and the right to unionize and negotiate collective bargains

20. L Elisa Celis, Sai Praneeth Reddy, et al., « Assignment Techniques for Crowdsourcing Sensitive Tasks », *in*: *Proc. of CSCW'16*, 2016, pp. 836–847.

21. Hiroshi Kajino, Yukino Baba, and Hisashi Kashima, « Instance-Privacy Preserving Crowdsourcing », *in*: *Proc. of HCOMP'14*, 2014.

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

23. Hiroshi Kajino, « Privacy-Preserving Crowdsourcing », PhD thesis, Univ. of Tokyo, 2015.

24. Hien To, Gabriel Ghinita, and Cyrus Shahabi, « A framework for protecting worker location privacy in spatial crowdsourcing », *in*: *Proc. of the VLDB Endow.* 7.*10* (2014), pp. 919–930.

25. Yuan Lu, Qiang Tang, and Guiling Wang, « Zebralancer: Private and anonymous crowdsourcing system atop open blockchain », *in*: *Proc. of ICDCS'18*, IEEE, 2018, pp. 853–865.

26. `https://www.youtube.com/watch?v=ggigQxtN5vU`

27. *Regulating gig, crowd, and platform work*, URL: `https://humancomputerinteraction.wvu.edu/crowd-work/literature-review`.

over pay and working conditions with their employer. A worker who is classified as self employed has none of these rights.".

Although these lighter protections are sources of conflict, rare are the situations in which workers are legally considered as employees in crowdsourcing platforms: even when courts conclude that workers should legally be employees (and not contractors), crowdsourcing companies do their best to limit the decision to very specific cases [28]. This conflict over the status of workers can be a source of instability for both the platforms and the workers, but also explains why more and more media address this topic. It is yet noticeable that, although profitability of crowdsourcing platforms heavily depends on this status, it is highly unlikely that forbidding the contractor status would end all these businesses [29].

However, the conflict about the status of workers is not the only issue reported in the news. Privacy issues are also reported, as trade-unionists were allegedly tracked by Deliveroo [30], and tracking may be an issue with Uber too, as it was allegedly used to track a journalist during an investigation [31]. Other examples also include cases where workers' personally identifiable information is trivially exposed online [32].

1.2.2 Crowdsourcing in Social Sciences

These questions, prominent in the media, are also studied in Social Sciences. We here present some findings and views of crowdsourcing that have risen from Social Sciences works, together with some political issues [33].

28. Lawsuits exist since 2015, in both California and Florida, and yet, Uber still tries to avoid these rulings. In France, similar decisions can be seen.
`https://www.nytimes.com/2015/06/18/business/uber-contests-california-labor-ruling-that-says-drivers-should-be-employees.html`
`https://www.theguardian.com/technology/2020/feb/07/uber-ab5-changes-drivers-california`
`https://www.lexpress.fr/actualite/societe/justice/la-cour-de-cassation-requalifie-en-contrat-de-travail-le-lien-entre-uber-et-un-chauffeur_2120045.html`

29. According to Uber, Deliveroo and Amazon
`https://www.theguardian.com/business/2017/feb/22/amazon-deliveroo-uber-hermes-still-viable-no-gig-economy-workers`

30. Internal emails that were leaked from Deliveroo indicate that the geolocation system of Deliveroo was used internally for identifying the riders that participated to strikes against the platform
`https://www.lemonde.fr/culture/article/2019/09/24/television-cash-investigation-a-la-rencontre-des-nouveaux-proletaires-du-web_6012758_3246.html`

31. `https://www.theverge.com/2014/11/19/7245447/uber-allegedly-tracked-journalist-with-internal-tool-called-god-view`

32. Matthew Lease, Jessica Hullman, et al., « Mechanical Turk is Not Anonymous », *in*: *SSRN Electronic Journal* (2013).

33. Although we hope not to misrepresent these fields or their findings, we do not pretend that this selection of results is fully representative, and mostly focus on criticisms that guide and motivate our

From this perspective, some studies provide interesting results on the population that constitutes the workforce of crowdsourcing: for instance, it has been shown [34] that, for French workers of crowdsourcing platforms, the most important motivation is money, and that 41% of them do not benefit from regular pension or wage. The social impact of crowdsourcing is firmly criticized by Anwar et al. [35], which argue that "gig work in its current capitalist manifestation can create and maintain conditions of precarity and vulnerability among gig workers in Africa", as the "structural and technological design of platforms contributes towards loneliness and social isolation, high work intensity, non-payment of wages, and unfair dismissals which adds to African workers' precariousness and vulnerability".

In a wider perspective, the analysis from Schmidt [36] insists on the key role of surveillance in crowdsourcing, taking the example of Upwork, to assert that "the freelance marketplaces are characterised by a relatively high level of surveillance. Upwork, for example, uses a software application called "Work Diary" to allow clients to virtually look over the shoulders of their independent contractors. Six times per hour and at random intervals, the software takes screenshots of the freelancers' computer.". A focus is given by Casilli [37] on the importance of recognizing crowdsourcing as work: this work should not be disregarded nor workers be dehumanized, and a better regulation is required. In this matter, the work of De Stefano [38] mentions a "cultural struggle to avoid that workers are perceived as extensions of platforms, apps, and IT-devices", a problematic perception which seems to be encouraged even in some academic Computer Science articles, such as Parameswaran et al. [39], who propose a system that aims at "[appearing] to the end user as similar as possible to a conventional database system (a relational one in our case), while hiding many of the complexities of dealing with humans as data sources". As for regulation, De Stefano [40] also reminds us that "it is far from being demonstrated that deregulation of

work.

34. Antonio Casilli, Paola Tubaro, et al., « Le Micro-Travail en France. Derrière l'automatisation, de nouvelles précarités au travail? », *in*: (2019).

35. Mohammad Amir Anwar and Mark Graham, « Between a rock and a hard place: Freedom, flexibility, precarity and vulnerability in the gig economy in Africa », *in*: *Competition & Change* (2019), p. 1024529420914473.

36. Florian Alexander Schmidt, *Crowd Design: From Tools for Empowerment to Platform Capitalism*, Birkhäuser, 2017.

37. Casilli, *op. cit.*

38. De Stefano, *op. cit.*

39. Aditya Ganesh Parameswaran, Hyunjung Park, et al., « Deco: declarative crowdsourcing », *in*: *Proc. of CIKM'21*, 2012, pp. 1203–1212.

40. De Stefano, *op. cit.*

labor markets and of nonstandard forms of work in particular has positive impacts on growth, innovation or employment rates."

As for solutions, many suggestions exist: technical suggestions include ensuring transparency and fairness in all business decisions, such as ratings, deactivation of profiles, or change in the conditions of use [41]. Portability of profiles, and ratings in particular, is also suggested. However, most propositions include political, legal or economic rethinking of crowdsourcing. For instance, Casilli [42] proposes three main solutions: giving the employee status to workers when it is possible, helping the development of non-profit platforms, and giving to personal data a common good status, which is still to be defined with more details, depending on the nature of tasks.

1.3 Our answers

In this work, we focus on two aspects: privacy and regulation. First privacy, because we believe that many issues are possible due to surveillance from requesters or platforms. Then regulation and transparency, to make it possible for law-makers to enforce their decisions, and end-users to prove their cases in courts. Due to the diversity of studied domains, and to preserve clarity, we expose in each chapter the corresponding state of the art.

As surveillance is a key issue in crowdsourcing, we believe that this protection cannot be achieved without preventing platforms, requesters, or even other participants from obtaining private information. However, as private information about workers is also used for assigning relevant tasks to workers, or to retrieve meaningful statistics for the platform, it is not sufficient to simply evacuate them. For instance, the two critical aspects highlighted by Schmidt [43] to characterize digital labour platforms (mostly focusing on freelance marketplaces) are (1) the impact of skills and specialization of workers on the stability of their job and on their revenue, and (2) the high level of surveillance to which workers are submitted. In Chapter 2, we propose two main contributions that use privacy-preserving techniques along with encryption tools. First, we compute privacy-preserving statistics on the worker population with an algorithm we developed: the PKD algorithm. These statistics allow the platform to quantify the available workforce more precisely, either to advertise it to requesters, or to help requesters build more suited tasks by

41. *Ibid.*
42. Casilli, *op. cit.*
43. Schmidt, *op. cit.*

adapting them to the crowd. A demonstration has been developed, to illustrate a possible use-case of these statistics. Security proofs and experiments on quality and computation time are also provided to validate our approach. Then, these statistics are used to assign tasks to workers, by making use of Private Information Retrieval (PIR) techniques in order to submit relevant tasks to workers. As these techniques are not designed to be used multiple times, proper adaptations are made to preserve their protective properties. This contribution is also validated by security proofs and experiments on quality and computation time. This work has led to several publications [44] [45] [46] [47] [48] [49] [50].

However, breaking surveillance is not the only way to protect workers. As seen in Section 1.2, regulations and politics are demanded by workers, and law-makers' attention is drawn to crowdsourcing accordingly. In this context, it seems natural to study how technical tools could be built to support possible legislations. In Chapter 3, we present SEPAR, a system that makes it possible to both enforce restrictions on activities and prove participations in crowdsourcing processes. This system considers a multi-platform setting to which any platform can be plugged, and restricts the impact on privacy as much as possible. That way, we aim at providing technical solutions to enforce limits on work time that would take a whole crowdsourcing ecosystem into account, and to make it possible for users to prove their participation in tasks they did (*e.g.* for engaging lawsuits against abusers or in order to prove their eligibility to social grants). This work will soon be submitted for a publication in the SIGMOD conference [51].

In Chapter 4, we extend our work on PIR techniques from Chapter 2, by generalizing it to cope with other contexts than crowdsourcing. Indeed, the limitations of PIR when multiple downloads are performed are not specific to crowdsourcing, and can also be seen in other contexts such as media consumption [52]. This chapter presents our questions and findings on this topic, with both an attack model and analyses of the possible countermeasures. However, we stress that this work is still ongoing, and has not reached yet the

44. Duguépéroux and Allard, « From Task Tuning to Task Assignment in Privacy-Preserving Crowdsourcing Platforms ».
45. *Idem*, « Privacy-Preserving Informed Task Design in Crowdsourcing Processes ».
46. Duguépéroux, Voyez, and Allard, « Task-Tuning in Privacy-Preserving Crowdsourcing Platforms ».
47. Duguépéroux, « Guaranteed Confidentiality and Efficiency in Crowdsourcing Platforms ».
48. *Idem*, « Guaranteed Confidentiality and Efficiency in Crowdsourcing Platforms ».
49. Duguépéroux and Allard, « Un algorithme de partitionnement de l'espace des compétences pour plateformes de crowdsourcing respectueuses de la vie privée ».
50. Duguépéroux, Voyez, and Allard, « Task-Tuning in Privacy-Preserving Crowdsourcing Platforms ».
51. Amiri, Duguépéroux, et al., *op. cit.*
52. Trinabh Gupta, Natacha Crooks, et al., « Scalable and Private Media Consumption with Popcorn. », *in*: *Proc. of NSDI'16*, 2016, pp. 91–107.

same maturity as the others.

Finally, in Chapter 5, we conclude by providing first an overview of our work and then our own point of view on possible future work to continue our contributions.

Chapter 2

From task tuning to task assignment in privacy-preserving crowdsourcing platforms

2.1 Introduction

Crowdsourcing platforms are online intermediates between *requesters* and *workers*. The former have *tasks* to propose to the latter, while the latter have *profiles* (*e.g.,* skills, devices, experience, availabilities) to propose to the former. Crowdsourcing platforms have grown in diversity, covering application domains ranging from micro-tasks [1] or home-cleaning [2] to collaborative engineering [3] or specialized software team design [4].

The efficiency of crowdsourcing platforms especially relies on the wealth of information available in the profiles of registered workers. Depending on the platform, 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.* This holds especially for platforms dedicated to specialized tasks that require strongly qualified workers. But even micro-tasks platforms may maintain detailed worker profiles (see, *e.g.,* the *qualification* system of *Amazon Mechanical Turk* that maintains so-called *premium qualifications* [5] - *i.e.,* sociodemographic information such as *age range*, *gender*, *employment*, *marital status*, *etc.* - in the profiles of workers willing to participate to surveys). The availability of such detailed worker profiles is of utmost importance to both requesters and platforms because it enables:

1. https://www.mturk.com/
2. https://www.handy.com/
3. https://www.kicklox.com/
4. https://tara.ai/
5. https://requester.mturk.com/pricing

Primary usages of worker profiles: to target the specific set of workers relevant for a given task (through *e.g.,* elaborate task assignment algorithms [6]).

Secondary usages of worker profiles: to describe the population of workers available, often through `COUNT` aggregates, in order, for instance, to promote the platform by ensuring requesters that workers relevant for their tasks are registered on the platform [7], or to participate to the task design by letting requesters fine-tune the tasks according to the actual population of workers (*e.g.,* setting wages according to the rarity of the skills required, adapting slightly the requirements according to the skills available).

Both primary and secondary usages are complementary and usually supported by today's crowdsourcing platforms, in particular by platforms dedicated to highly skilled tasks and workers [8].

However, the downside of fine-grained worker profiles is that detailed information related to personal skills can be highly identifying (*e.g.,* typically a unique combination of location/skills/centers of interest) or sensitive (*e.g.,* costly devices or high minimum wages may correspond to a wealthy individual, various personal traits may be inferred from centers of interest [9]). Recent privacy scandals, mentioned in the introduction of the manuscript, have shown that crowdsourcing platforms are not immune to negligences or misbehaviours. It is noticeable that workers nevertheless expect platforms to secure their data and to protect their privacy in order to lower the privacy threats they face [10]. Moreover, in a legal context where laws firmly require businesses and public organizations to safeguard the privacy of individuals (such as the European GDPR [11] or the California Consumer Privacy Act [12]), legal compliance is also a strong incentive for platforms for designing and implementing sound privacy-preserving crowdsourcing processes. Ethics

6. Panagiotis Mavridis, David Gross-Amblard, and Zoltán Miklós, « Using Hierarchical Skills for Optimized Task Assignment in Knowledge-Intensive Crowdsourcing », *in*: *Proc. of WWW'16*, 2016, pp. 843–853.

7. See for example the Kicklox search form (`https://www.kicklox.com/en/`) that inputs a list of keywords (typically skills) and displays the corresponding number of workers available.

8. See for example, Kicklox (`https://www.kicklox.com/en/`) or Tara (`https://tara.ai/`). The secondary usage consisting in promoting the platform is sometimes performed through a public access to detailed parts of worker profiles (*e.g.,* Malt (`https://www.malt.com/`), 404works (`https://www.404works.com/en/freelancers`)).

9. See, *e.g.,* `http://applymagicsauce.com/about-us`

10. Huichuan Xia, Yang Wang, et al., « Our Privacy Needs to be Protected at All Costs: Crowd Workers' Privacy Experiences on Amazon Mechanical Turk », *in*: *Proc. of HCI'17* 1 (2017), p. 113.

11. `https://eur-lex.europa.eu/eli/reg/2016/679/oj`

12. `https://www.caprivacy.org/`

in general, and privacy in particular, are indeed clearly identified as key issues for next generation *future of work* platforms [13]. Most related privacy-preserving works have focused on the primary usage of worker profiles, *i.e.,* the task-assignment problem (*e.g.,* based on additively-homomorphic encryption [14] or on local differential privacy [15] [16]).

Our goal in this chapter is twofold: (1) *consider both primary and secondary usages as first-class citizens* by proposing a privacy-preserving solution for computing multi-dimensional `COUNTs` over worker profiles and a task assignment algorithm based on recent affordable Private Information Retrieval (PIR) techniques, and (2) *integrate well with other privacy-preserving algorithms possibly executed by a platform* without jeopardizing the privacy guarantees by requiring our privacy model to be composable both with usual computational cryptographic guarantees provided by real-life encryption schemes and with classical differential privacy guarantees as well.

These two problems are not trivial. First, we focus on secondary usages. The problem of computing multi-dimensional `COUNTs` over distributed worker profiles in a privacy-preserving manner is not trivial. Interactive approaches - that issue a privacy-preserving `COUNT` query over the set of workers each time needed (*e.g.,* a requester estimates the number of workers qualified for a given task) - are inadequate because the number of queries would be unbounded. This would lead to out-of-control information disclosure through the sequence of `COUNTs` computed [17] [18]. Non-interactive approaches are a promising avenue because they compute, once for all and in a privacy-preserving manner, the static data structures which are then exported and queried by the untrusted parties (*e.g.,* platform, requesters) without any limit on the number of queries. More precisely, on the one hand, hierarchies of histograms are well-known data structures that support `COUNT` queries and that cope well with the stringent privacy guarantees of differential privacy [19]. However, they

13. participants, *op. cit.*
14. Kajino, *op. cit.*
15. Béziaud, Allard, and Gross-Amblard, *op. cit.*
16. Note that limiting the information disclosed to the platform (*i.e.,* perturbed information about worker profiles) relieves platforms from the costly task of handling personal data. The European GDPR indeed explicitely excludes anonymized data from its scope (see Article 4, Recital 26 `https://gdpr-info.eu/recitals/no-26/`).
17. Aloni Cohen and Kobbi Nissim, « Linear Program Reconstruction in Practice », *in*: *CoRR* (2018), arXiv: `1810.05692`.
18. Irit Dinur and Kobbi Nissim, « Revealing information while preserving privacy », *in*: *Proc. of SIGACT-SIGMOD-SIGART'03*, 2003, pp. 202–210.
19. Wahbeh Qardaji, Weining Yang, and Ninghui Li, « Understanding Hierarchical Methods for Differentially Private Histograms », *in*: *Proc. VLDB Endow.* 6.*14* (2013), pp. 1954–1965, ISSN: 2150-8097.

do not cope well with more than a few dimensions [20], whereas a worker profile may contain more skills (*e.g.,* a dozen), and they require a trusted centralized platform. On the other hand, privacy-preserving spatial decompositions [21] [22] are more tolerant to a higher number of dimensions but require as well a trusted centralized platform. Second, algorithms for assigning tasks to workers while providing sound privacy guarantees have been proposed as alternatives against naive *spamming approaches* - where all tasks are sent to all workers. However they are either based (1) on perturbation only [23] and suffer from a severe drop in quality or (2) on encryption only [24] but they do not reach realistic performances, or (3) they focus on the specific context of spatial crowdsourcing and geolocation data [25].

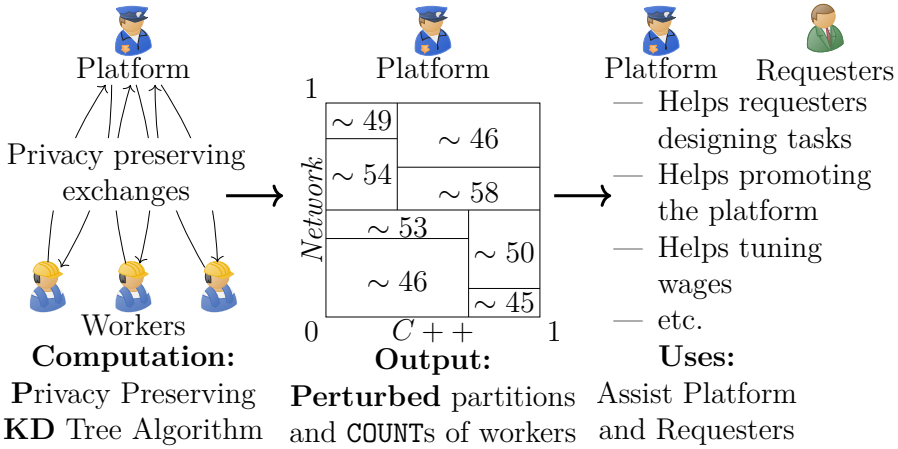


Figure 2.1 – Overview of the PKD algorithm: supporting secondary usages of worker profiles with privacy guarantees

Our Contribution First, we propose to benefit from the best of the two non-interactive approaches described above by computing a privacy-preserving space partitioning of the worker profiles (for coping with their dimensionality) based on perturbed 1-dimensional histograms (for their nice tolerance to differentially private perturbations). We propose

20. *Ibid.*
21. Graham Cormode, Cecilia Procopiuc, et al., « Differentially private spatial decompositions », *in*: *Proc. of ICDE'12*, 2012, pp. 20–31.
22. Jun Zhang, Xiaokui Xiao, and Xing Xie, « PrivTree: A Differentially Private Algorithm for Hierarchical Decompositions », *in*: *Proc. of SIGMOD'16*, 2016, pp. 155–170.
23. Béziaud, Allard, and Gross-Amblard, *op. cit.*
24. Kajino, *op. cit.*
25. Hien To, Cyrus Shahabi, and Li Xiong, « Privacy-preserving online task assignment in spatial crowdsourcing with untrusted server », *in*: *Proc. of ICDE'18*, 2018, pp. 833–844.

the Privacy-preserving KD-Tree algorithm (PKD for short, depicted in Figure 2.1), a privacy-preserving algorithm for computing a (perturbed) multi-dimensional distribution of skills of the actual population of workers. The PKD algorithm is distributed between mutually distrustful workers and an untrusted platform. It consists in splitting recursively the space of skills in two around the median (similarly to the *KD-tree* construction algorithm) based on the 1-dimensional histogram of the dimension being split, and it protects workers' profiles all along the computation by combining additively-homomorphic encryption together with differentially private perturbation. No raw worker profile is ever communicated, neither to the platform nor to other workers. The output of the PKD algorithm is a hierarchical partitioning of the space of skills together with the (perturbed) COUNT of workers per partition (see Figure 2.1). The PKD algorithm is complementary to privacy-preserving task assignment works and can be used in conjunction with them provided that the privacy models compose well. In particular, since our privacy model is a computational variant of differential privacy, the PKD algorithm composes well with state-of-the-art approaches [26] [27] since they are based on usual computational cryptographic model or differential privacy model. Second, we use Private Information Retrieval techniques [28] in order to design a solution to task assignment that is both private and affordable. We perform an in-depth study of the problem of using PIR techniques for proposing tasks to workers, show that it is NP-Hard, and come up with the PKD PIR Packing heuristic that groups tasks together according to the partitioning output by the PKD algorithm. Obviously, the PKD PIR Packing heuristic composes well with the PKD algorithm.

More precisely, we make the following contributions:

1. We design the PKD algorithm, a distributed privacy-preserving algorithm for computing a multi-dimensional hierarchical partitioning of the space of skills within a population of workers.
2. We formally prove the security of the PKD algorithm against *honest-but-curious* attackers. The PKD algorithm is shown to satisfy a computational variant of differential privacy called the ϵ_κ-SIM-CDP model. We provide a theoretical analysis of its complexity.
3. We provide an in-depth study of the problem of using PIR techniques for proposing tasks to workers and design the PKD PIR Packing heuristic that benefits from the

26. Kajino, *op. cit.*
27. Béziaud, Allard, and Gross-Amblard, *op. cit.*
28. Carlos Aguilar-Melchor, Joris Barrier, et al., « XPIR: Private information retrieval for everyone », *in*: *Proc. of PET'16* 2016.*2* (2016), pp. 155–174.

partitioning computed by the `PKD` algorithm for grouping tasks together. We show that the `PKD PIR Packing` heuristic satisfies our privacy model.

4. We provide an extensive experimental evaluation of the `PKD` algorithm and of the `PKD PIR Packing` heuristic over synthetic and realistic data that demonstrates their quality and performance in various scenarios. Our realistic skills dataset is built from data dumps of *StackExchange* online forums.

Overall, this work has led to multiple publications [29] [30] [31] [32] [33] [34] [35].

The chapter is organized as follows. In Section 2.2, we survey the related work. Section 2.3 introduces the participant model, the security and privacy models, and the technical tools necessary in the rest of the chapter. Section 2.4 describes the `PKD` algorithm in details and formally analyzes its cost and security. Section 2.5 studies the problem of using PIR techniques for task assignment, describes the `PKD PIR Packing` heuristic, and formally analyzes its security. We discuss some details on how to allow updates for both the `PKD` algorithm and the `PKD PIR Packing` heuristic in Section 2.7. Section 2.6 experimentally validates their quality and efficiency. In Section 2.8, we provide a proof of concept for some of these techniques. Finally, Section 2.9 concludes and discusses interesting future works.

2.2 Related work

We present here works that focus on topics closely related to those presented in this chapter: privacy preserving space-partitioning and task-assignment, and task design.

2.2.1 Privacy-preserving space partitioning

We focus here on techniques which are not directly related with crowdsourcing, but provide helpful background to understand our use of private histograms and KD-trees. It is noticeable that these techniques, which are centralized and do not take into account some specificities of crowdsourcing, cannot be directly compared to our work. Yet, these

29. Duguépéroux and Allard, « From Task Tuning to Task Assignment in Privacy-Preserving Crowdsourcing Platforms ».
30. *Idem*, « Privacy-Preserving Informed Task Design in Crowdsourcing Processes ».
31. Duguépéroux, Voyez, and Allard, « Task-Tuning in Privacy-Preserving Crowdsourcing Platforms ».
32. Duguépéroux, « Guaranteed Confidentiality and Efficiency in Crowdsourcing Platforms ».
33. *Idem*, « Guaranteed Confidentiality and Efficiency in Crowdsourcing Platforms ».
34. Duguépéroux and Allard, « Un algorithme de partitionnement de l'espace des compétences pour plateformes de crowdsourcing respectueuses de la vie privée ».
35. Duguépéroux, Voyez, and Allard, « Task-Tuning in Privacy-Preserving Crowdsourcing Platforms ».

techniques provide interesting tools to privately estimate a distribution. We first introduce quickly a few elements on the interest of count queries in research, before focusing on histograms, and finally presenting more complex structures used in spatial partitioning. In this section, we make extensive use of differential privacy notations and concepts, and refer the unfamiliar reader to Section 2.3.2 for more details.

Private count queries One common way to use differential privacy is through the computation of counts, in order to compute the number of individuals in a database who match some criteria. Indeed, computing differentially private counts is relatively straightforward since the sensitivity of counts is 1: the difference between the counts of two databases differing by at most one individual cannot exceed 1. Therefore, a straightforward application of differential privacy is sufficient.

However, most applications (such as ours) require multiple counts. Although composability theorems offer interesting ways to deal with multiple count queries, the privacy budget consumed remains linear in the number of queries (unless these queries can be parallelized). Furthermore, if queries are not managed at all, redundant counts may be computed, which are likely to consume budget for a limited gain. Therefore, many works focus on finding more efficient ways to compute either significant counts, that can be used for multiple queries without increasing the privacy budget, or on optimal distribution of this budget throughout requests.

Private histograms As we used histograms in our approach, we present here some families of approaches that are dedicated to computing high quality histograms. More precisely, we focus here on histograms representing the whole distribution of a one-dimensional domain. Histograms are interesting as they are based on parallel computation (*i.e.* computation on disjoint sub-datasets according to differential privacy standards), and are therefore well-suited for differential privacy budget optimization. Intuitively, there are two main possible ways to compute an histogram. First, the one called *flat method*[36], directly computes the counts of all bins of the histogram. This method makes full use of the parallelism of the data structure, but its results may lack precision for counts that require more than one bin. The other method, called *hierarchical method*, uses a tree structure and combines both parallel and sequential compositions to make use of redundancies. One interesting optimization allowed by this structure has been introduced by Hay et

36. Qardaji, Yang, and Li, *op. cit.*

al. [37], and relies on the consistency of hierarchical histograms. Indeed, through hierarchical histograms, multiple queries appear to be redundant. For instance, if we compute an histogram of the number of workers according to their age, we may learn that there are, in the dataset, 37 workers between 20 and 40 years old, 25 workers between 40 and 60 years old, etc. Due to the hierarchical structure, we may also learn information on more precise ranges, *e.g.* 20 workers between 20 and 30 years old, and 12 workers between 30 and 40 years old. Note that these values are not necessarily consistent with each other, due to the noise induced by differential privacy. The idea of Hay et al. [38] is to make use of the knowledge that underlying data is consistent, to improve the overall quality of results. In that case, the sum of workers between 20 and 30 and between 30 and 40 should equal the number of workers between 20 and 40. As the noise distribution is centered on 0, it is possible to use this knowledge to reduce the error (*e.g.* the real number of workers between 20 and 40 is likely to be between 32 and 34). In a similar idea, Qardaji et al. [39] propose optimizations by showing how to select the best branching factor (the numbeer of children of each node in a tree) while taking this consistency into account.

Private spatial partitioning These techniques are useful for histograms, but can also be used in another context: spatial partitioning. Indeed, an histogram is actually a partitioning of a one-dimensional space. From this perspective, the extension to 2 or more dimensional spaces seems quite natural. An important improvement brought by Cormode et al. [40], consists in optimizing the ϵ budget distribution policy among the levels of the hierarchy. Although it would be intuitive to split the ϵ budget equally among all depths, it is more efficient to use a *geometric* policy, with a budget proportional to $2^{i/3}$ for a depth i ($i = 0$ for the root of the hierarchy). Another possible optimization proposed by Qardaji et al. [41] introduces *adaptative grids*, that compute finer grain for bins with higher density.

Although they are a bit out of our focus, we can also mention other works that focus on improving the computation of differentially private hierarchical histograms. Indeed, some works propose to eliminate the need of fixing the height of trees beforehand (which can be hard to parametrize although it has an important impact on computation time

37. Michael Hay, Vibhor Rastogi, et al., « Boosting the accuracy of differentially private histograms through consistency », *in*: *Proc. of the VLDB Endow.* 3.*1-2* (2010), pp. 1021–1032.
38. *Ibid.*
39. Qardaji, Yang, and Li, *op. cit.*
40. Cormode, Procopiuc, et al., *op. cit.*
41. Wahbeh Qardaji, Weining Yang, and Ninghui Li, « Differentially private grids for geospatial data », *in*: *Proc. of ICDE'13*, 2013, pp. 757–768.

and precision), and replace it with other parameters easier to manage [42], to tackle the efficiency issues of privacy-preserving hierarchies of histograms [43], or to apply a general benchmark method to compare histograms [44].

2.2.2 Privacy-preserving task assignment

Recent works considering privacy issues in crowdsourcing have focused a lot on the privacy-preserving task assignment problem. Literature on this topic can be split into two main categories. First, what could be called *general* crowdsourcing (general meaning that it does not focus on a specific application of crowdsourcing), and second *spatial* crowdsourcing, which refers to crowdsourcing applications that make use of geolocation, such as Uber, Deliveroo, taking pictures at some specific places, etc. We present now the main advances within each of these categories, along with their weaknesses, which we attend to solve in our work. In this section, we also make extensive use of differential privacy notations and concepts, which details and definitions are presented in Section 2.3.2.

General crowdsourcing In general crowdsourcing, several works exist that focus on privacy-preserving task assignment.

First, Kajino [45] proposes an approach that makes extensive use of additively-homomorphic encryption. This work is interesting as the quality of the assignment is optimal, does not suffer from any information loss, and preserves privacy without requiring a trusted third party. However, their protocol uses homomorphic encryption a lot, which results in a prohibitive cost in terms of performance, unusable in real-life scenarios (*e.g.* 4.5 days for computing the assignment of 100 workers and tasks after optimization according to the author).

Another approach, relying on differential privacy, is proposed by Béziaud et al. [46]. Each worker profile - a vector of bits - is perturbed locally by the corresponding worker, based on a local differentially private bit flipping scheme [47]. A classical task-assignment

42. Zhang, Xiao, and Xie, *op. cit.*
43. Georgios Kellaris, Stavros Papadopoulos, and Dimitris Papadias, « Engineering Methods for Differentially Private Histograms: Efficiency Beyond Utility », *in*: *IEEE TKDE* 31.*2* (2018), pp. 315–328.
44. Michael Hay, Ashwin Machanavajjhala, et al., « Principled evaluation of differentially private algorithms using dpbench », *in*: *Proc. of SIGMOD'16*, ACM, 2016, pp. 139–154.
45. Kajino, *op. cit.*
46. Béziaud, Allard, and Gross-Amblard, *op. cit.*
47. Cynthia Dwork and Aaron Roth, « The algorithmic foundations of differential privacy », *in*: *Foundations and Trends in Theoretical Computer Science* 9.*3–4* (2014), pp. 211–407.

algorithm can then be launched on the perturbed profiles and the tasks. The application of noise locally, by each worker, makes it possible to avoid heavy encryption schemes, as all transmitted data is already secured. Furthermore, it is noticeable that this procedure is possible even if the platform does not collaborate (the worker would then be protected, but the matching may be deteriorated). However, local perturbation also has its limits. As the perturbation is applied locally, and multiple times, it appears to heavily deteriorate the quality of the assignment: compared to a centralized context (with a trusted party adding noise only once), the perturbation required leads to more noise in the result [48]. The lack of precision that derives from it makes it hard to rely on in a real-life scenario as well.

Spatial crowdsourcing Other works have focused on the specific context of spatial crowdsourcing. Similarly to general crowdsourcing, spatial crowdsourcing aims at optimizing the assignment of workers to tasks. However, some specificities of spatial crowdsourcing make the proposed solutions inadequate in a more general context. For instance, these specificities include a focus on a small number of dimensions (typically, the two dimensions of a geolocation) and often an incompatibility with static worker profiles: indeed, spatial crowdsourcing mostly consider mobile and dynamic profiles, for which revealing perturbed information at multiple times can be considered less dangerous than doing the same with static profiles (for which these multiple perturbed divulgations make predictions more and more precise). However, this context also enables solutions that make use of these specificities (that cannot be assumed in a general crowdsourcing context), often resulting in a better quality and/or computation time.

For instance, To et al. [49] assume that (1) workers use a cellphone, and (2) geolocation is already known by the *cellular service provider* (*e.g.* through cell tower triangulation). Therefore, this service provider can be used as a trusted third party to perturb geolocation of workers in order to release differentially private data to crowdsourcing platforms.

Other works [50] [51] use a definition derived from differential privacy called *geo-indistinguishability*. As opposed to differential privacy, which is more suited for aggregates and hiding individuals in a crowd, geo-indistinguishability makes the privacy level depend on the position and neighborhood. To summarize, a probabilistic function fulfilling geo-

48. *Ibid.*
49. To, Ghinita, and Shahabi, *op. cit.*
50. To, Shahabi, and Xiong, *op. cit.*
51. Miguel E Andrés, Nicolás E Bordenabe, et al., « Geo-indistinguishability: Differential privacy for location-based systems », *in*: *Proc. of SIGSAC CCS'13*, 2013, pp. 901–914.

indistinguishability guarantees that close individuals are nearly indistinguishable from each other (meaning that the distributions of possible outputs are close), but this guarantee diminishes with the distance. This definition makes it possible to obtain interesting results without compromising privacy too much [52]. Yet, it is noticeable that this notion is less protective than differential privacy: for instance, as the guarantee diminishes with the distance, an isolated individual might be quickly singled out using geo-indistinguishable functions, which is not possible with differentially private functions.

For Zhai et al. [53], the privacy guarantee does not come from differential privacy or any anonymization processes based on data perturbation, but from another family of anonymization techniques: k-anonymity [54]. A k-anonymized dataset is a dataset in which any individual is indistinguishable from at least $k-1$ other individuals. Although this notion is often lighter and easier to use than differential privacy, many limits or flaws have been revealed along the years [55] [56] [57] [58] [59] [60] [61] (*e.g.* issues when multiple k-anonymized datasets are released, when considering external knowledge of the attacker, etc.). This work is still interesting in the way it combines many different techniques, from auction mechanisms, to oblivious transfer, homomorphically encrypted XOR (exclusive or) and k-anonymity, to achieve private assignment for spatial crowdsourcing.

Finally, Tao et al. [62] use geo-indistinguishability combined with a tree structure. The idea is to build a tree in which leaves match some areas of the space (such that the whole space considered is mapped with the tree). After that, rather than adding noise to perturb

52. To, Shahabi, and Xiong, *op. cit.*

53. Dongjun Zhai, Yue Sun, et al., « Towards secure and truthful task assignment in spatial crowdsourcing », *in*: *World Wide Web* 22.*5* (2019), pp. 2017–2040.

54. Latanya Sweeney, « k-anonymity: A model for protecting privacy », *in*: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10.*05* (2002), pp. 557–570.

55. Bee-Chung Chen, Kristen LeFevre, and Raghu Ramakrishnan, « Privacy skyline: privacy with multidimensional adversarial knowledge », *in*: *Proc. of VLDB'07*, VLDB Endowment, 2007, pp. 770–781.

56. Daniel Kifer, « Attacks on privacy and deFinetti's theorem », *in*: *Proc. of SIGMOD'09*, 2009, pp. 127–138.

57. Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian, « t-closeness: Privacy beyond k-anonymity and l-diversity », *in*: *Proc. of ICDE'07*, IEEE, 2007, pp. 106–115.

58. Ashwin Machanavajjhala, Johannes Gehrke, and Michaela Götz, « Data publishing against realistic adversaries », *in*: *Proc. of VLDB'07* 2.*1* (2009), pp. 790–801.

59. David J Martin, Daniel Kifer, et al., « Worst-case background knowledge for privacy-preserving data publishing », *in*: *Proc. of ICDE'07*, IEEE, 2007, pp. 126–135.

60. Raymond Chi-Wing Wong, Ada Wai-Chee Fu, et al., « Minimality attack in privacy preserving data publishing », *in*: *Proc. of VLDB'07*, 2007, pp. 543–554.

61. Xiaokui Xiao and Yufei Tao, « M-invariance: towards privacy preserving re-publication of dynamic datasets », *in*: *Proc. of SIGMOD'07*, 2007, pp. 689–700.

62. Qian Tao, Yongxin Tong, et al., « Differentially Private Online Task Assignment in Spatial Crowdsourcing: A Tree-based Approach », *in*: *Proc. of ICDE'20*, IEEE, 2020, pp. 517–528.

geolocation, for a given position, the revealed obfuscated position will be computed as follows. First, consider the leaf of the real position. Then, go upward the tree, step after step, with a given probability depending mostly on the arity of the tree and on ϵ. After that, go downwards, down to a leaf, choosing the child uniformly and randomly at each step. The arrival position will be revealed. The overall mechanism is proven to enforce geo-indistinguishability, and the quality of this approach overcomes previously found results with similar guarantees.

All these works are interesting in many ways, but most are confronted to some limitations. First, as they focus on spatial crowdsourcing, they cannot be applied to other contexts: their algorithms may not work as efficiently with higher dimensional spaces. Then, the security guarantees provided by these algorithms does not protect as well as differential privacy, and isolated individuals (in the considered space) may suffer from it. Finally, as they focus on spatial issues, and as locations may change with time, these algorithms require to be used frequently, which may lead to issues for static profiles, whether they rely on k-anonymity (which does not compose well with itself [63]), or on geo-indistinguishability (as it increases the ϵ budget without upper bound).

2.2.3 Task design

To the best of our knowledge, the problem of designing a task according to the actual crowd while providing sound privacy guarantees has not been studied by related works. Most works dealing with task design focus on the complexity of the task [64], on the interface with the worker [65 66 67], on the design of workflows [68 69], or on the filters that may be embedded within tasks and based on which relevant workers should be selected [70]. However, these approaches ignore the relevance of tasks with respect to the actual crowd, and thus

63. Xiao and Tao, *op. cit.*

64. Ailbhe Finnerty, Pavel Kucherbaev, et al., « Keep it Simple: Reward and Task Design in Crowdsourcing », *in*: *Proc. of SIGCHI'13*, 2013, 14:1–14:4.

65. Li, Wang, et al., *op. cit.*

66. Finnerty, Kucherbaev, et al., *op. cit.*

67. Pavel Kucherbaev, Florian Daniel, et al., « Crowdsourcing processes: A survey of approaches and opportunities », *in*: *IEEE Internet Computing* 20.*2* (2015), pp. 50–56.

68. Anand Pramod Kulkarni, Matthew Can, and Bjoern Hartmann, « Turkomatic: automatic, recursive task and workflow design for mechanical turk », *in*: *Proc. HCOMP'11*, 2011.

69. Anand Kulkarni, Matthew Can, and Björn Hartmann, « Collaboratively crowdsourcing workflows with turkomatic », *in*: *Proc. of CSCW'12*, 2012, pp. 1003–1012.

70. Mohammad Allahbakhsh, Boualem Benatallah, et al., « Quality control in crowdsourcing systems: Issues and directions », *in*: *IEEE Internet Computing* 17.*2* (2013), pp. 76–81.

ignore the related privacy issues.

2.3 Preliminaries

2.3.1 Participants model

Three types of participants collaborate together during our crowdsourcing process. *Workers* are interested in solving tasks that are relevant to their profiles; *requesters* propose tasks to be solved by relevant workers; and the *platform* is an intermediary between workers and requesters.

A worker profile $p_i \in \mathcal{P}$ is represented by an n-dimensional vector of floats, where each float value $p_i[j] \in [0, 1]$ represents the degree of competency of the worker i with respect to the jth skill. The set of skills available and their indexes within workers' profiles is static and identical for all profiles.

A task $t_k \in \mathcal{T}$ is made of two parts. First, the *metadata* part is a precise description of the worker profiles that are needed for the task completion. More precisely it is an n-dimensional subspace of the space of skills. This work does not put any constraint on the kind of subspace described in the metadata part (*e.g.,* hyper-rectangles, hyper-spheres, arbitrary set operators between subspaces). However, for the sake of concreteness, we focus below on metadata expressed as hyper-rectangles. More formally, the metadata $m_k \in \mathcal{M}$ of a task $t_k \in \mathcal{T}$ is an n-dimensional vector of ranges over skills where the logical connector between any pair of ranges is the conjunction. We call $m_k[j]$ the range of float values (between 0 and 1) for task k and skill j. The second part of a task consists in the necessary information for performing the task and is represented as an arbitrary bitstring $\{0, 1\}^*$. In this work, we essentially focus on the metadata part of tasks. We say that a worker and a task match if the point described by the worker profile belongs to the subspace described by the task metadata, *i.e.,* worker p_i and task t_k match if and only if $\forall j \in [0, n-1]$, then $p_i[j] \in m_k[j]$.

We do not make strong assumptions on the resources offered by participants. Workers, requesters, and the platform 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 - contrary to workers or requesters - similarly to a traditional client/server setting.

We assume that all participants follow the *honest-but-curious* attack model in that

they do not deviate from the protocol but make use of the information disclosed, in any computationally-feasible way, for inferring personal data. Workers may collude together up to a reasonable bound denoted by τ in the following.

2.3.2 Privacy tools

Our proposal builds on two families of protection mechanisms: a *semantically secure* encryption scheme and a *differentially private* perturbation scheme. The resulting overall privacy model thus integrates both families of guarantees together.

We explain here these two models, together with some related techniques that we use in this work.

Cryptography Cryptography originally aims at providing secure communications between two or more participants. Usually, two main families are distinguished: symmetric cryptography, that relies on a shared secret between participants in order to encode and decode communications, and asymmetric cryptography, that provides ways to avoid this shared secret, at the expanse of longer computation times. As we do not need symmetric cryptography background, we focus here on asymmetric cryptography.

Asymmetric cryptography, sometimes called public-key cryptography, uses pairs of keys to protect communications. First, a private key, that is not supposed to be shared with anyone, and then, a public key, that can be shared with anyone (and in particular, with the sender or receiver). The public key of a party (let say Alice) is used by another party (Bob) to send a protected message. For instance, Bob will use Alice's public key to encrypt a given message (anyone can send private messages), such that only Alice can decrypt this message thanks to her private key (only Alice can read her messages). On the other hand, the private key is used to decrypt the message. Asymmetric cryptography can also be used to create signatures. For instance, if Bob wants to prove to Alice that the message was really emitted by him, he can add a digital signature, created using his private key (therefore, only he can sign). Upon reception, Alice will then be able to verify Bob's signature by using his public key (anyone can check the signature).

Note that this whole scheme assumes a relationship between the public and the private key, but also that the private key cannot be deduced from the public key, nor from encrypted messages or signatures. These assumptions are legitimated through two main points. First, some theoretical results are assumed to be true, that ensure that finding the private key, to decrypt a message without it, or to forge a falsified signature is *hard*

(*e.g.* in most cases the computation time to do it is exponential in the size of the key), while doing it while knowing the private key is not. Then, it is assumed that the attacker has access to limited amount of time and computation power. *Semantic security*[71] is a security definition against such an attacker, whose computational power is bounded.

An usual way to prove security in cryptography is to use the simulation paradigm[72]. The idea is first to define an ideal behavior, that corresponds to what we want to achieve (*e.g.* no-one can read the message that Bob sent to Alice, and Alice knows that Bob's signature is valid), and then to prove that the protocol actually simulates this behavior. For instance, to protect a secret message, or a set of secrets, we define these sets, different parties (*e.g.* Alice, and an external attacker), and what they may learn. Then, we prove that, assuming a given set of hypothesis (limited computation time and power), whatever an attacker can gain through any attack using messages or other elements of the protocol, can also be obtained with a *benign behavior*, *i.e.* while conforming to the protocol. For instance, external attackers cannot learn anything about the message, Alice cannot learn anything more than the validity of Bob's signature and the content of the message (in particular, she cannot learn the private key of Bob).

Homomorphic encryption Homomorphic encryption is a form of encryption that allows useful computation to be made using only ciphertexts. With schemes allowing it, the idea is that the output of computations made on ciphertexts, when decrypted, is the result of some given operations (*e.g.* sums or multiplications) on decrypted ciphertexts given as inputs. For instance, with additively homomorphic encryption schemes[73], it is possible to take two ciphertexts as an input, and to output the encrypted sum of the cleartexts corresponding to the input (note that neither the keys nor the knowledge of the initial cleartexts is required in this algorithm, and that the result is also encrypted). Informally, given a and b two integers, $\mathtt{E}$ the encryption function, X the encryption key, K the decryption key, $\mathtt{D}$ the decryption function, and $+_h$ the homomorphic addition operator, then $\mathtt{D}_K(\mathtt{E}_X(a) +_h \mathtt{E}_X(b)) == a + b$.

This property is particularly interesting when using the storage and computing resources of an untrusted third party. For instance, it may be interesting to send encrypted values to

71. Shafi Goldwasser and Silvio Micali, « Probabilistic encryption », *in*: *Journal of computer and system sciences* 28.*2* (1984), pp. 270–299.

72. Oded Goldreich, « Foundations of Cryptography–A Primer », *in*: *Foundations and Trends® in Theoretical Computer Science* 1.*1* (2005), pp. 1–116.

73. Pascal Paillier, « Public-key cryptosystems based on composite degree residuosity classes », *in*: *Proc. of EUROCRYPT'99*, 1999, pp. 223–238.

a third party, that will perform computations over them according to a public algorithm, and outputs the encrypted result. The client can then decrypt the result while the third party will not learn anything about the data itself.

It is noticeable that this property has been widely studied, and that many homomorphic schemes exist. These schemes are implemented within various cryptographic schemes, and rely on diverse hypothesis (*e.g.* on the hardness of decomposing a number in prime factors, on finding the minimal length of a lattice vector, etc.). Furthermore, they also provide support for various operations. For instance, we can distinguish *fully homomorphic* encryption schemes [74] that provide arbitrary operations, *partially homomorphic* encryption schemes [75] that are limited to a given set of operations (either additions or multiplications), or *somewhat homomorphic* encryption schemes [76] that can perform arbitrary operations a limited number of times. However, even within this diversity, to the best of our knowledge, no solution exist that provides limitless number of additions and multiplications (for a so-called *fully homomorphic* encryption scheme) without ending in prohibitive computation time for real-life applications.

In this work, we benefit from partially homomorphic encryption schemes that satisfy the following properties. First, it must provide *semantic security guarantees*. Second, it must be *additively-homomorphic*: it must support additions. Third, the scheme must support *non-interactive threshold decryption*. We additionally use this last property, available in some schemes [77] [78]. It allows the decryption key to be *split* in n_K *key-shares* K_i such that a complete decryption requires to perform independently $T \leq n_K$ partial decryptions by distinct key-shares. Note that in a typical key generation setting, pairs of keys are generated once and for all by a non-colluding, independent entity.

The Damgard-Jurik cryptosystem [79], a generalization of Paillier's cryptosystem [80], is an instance of encryption scheme that provides the desired properties. We refer the interested reader to the original papers for details [81] [82].

74. Craig Gentry and Dan Boneh, *A fully homomorphic encryption scheme*, vol. 20, 9, Stanford university Stanford, 2009.
75. Paillier, *op. cit.*
76. Craig Gentry, « Fully homomorphic encryption using ideal lattices », *in*: *Proc. of STOC'09*, 2009, pp. 169–178.
77. Paillier, *op. cit.*
78. Ivan Damgård and Mads Jurik, « A generalisation, a simplification and some applications of paillier's probabilistic public-key system », *in*: *Proc. of PKC'01*, 2001, pp. 119–136.
79. *Ibid.*
80. Paillier, *op. cit.*
81. Damgård and Jurik, *op. cit.*
82. Paillier, *op. cit.*

Private Information Retrieval Private Information Retrieval (PIR) techniques allow a client to download binary objects called *items* (*e.g.,* a record, a movie) stored on a server in a *library* of objects, without revealing to the server which of the binary objects has been downloaded. We call this function the `PIR-get` function, and *request* its input. Emerging PIR protocols are now affordable and able to cope with the latency constraints of real-life scenarios (*e.g.,* in media consumption scenarios [83] [84] [85]). Our approach makes use of the security guarantees of PIR techniques. In this chapter, for concreteness, we consider a PIR protocol based on additively-homomorphic encryption called *XPIR* [86]. It is part of the *computational PIR* family of protocols, that provides semantic security guarantees.

It is to be noticed that three main parameters may affect computational PIR efficiency: (1) the size of the library itself, that has to be read for each call to the `PIR-get` function; (2) the size of items, that impacts the size of downloads (multiplied by an *expansion factor*, the ratio between the size of an encrypted value and the clear value); (3) the number of items (again, multiplied by the expansion factor), as it directly impacts the size of the request (which can be quite large in some cases).

As Chapter 4 focuses on PIR techniques and develops examples of PIR schemes, we do not go further into details here.

Differential privacy Differential privacy [87] [88] is a model that aims at providing a given protection to individuals within a database, while allowing statistical uses of the data it contains. As opposed to other models, such as k-anonymity [89], this model does not necessarily aim at publishing a database, but rather considers a function f, that takes the database as input, and publishes its anonymized output [90]. Although publishing an anonymized version of the whole database is hard, processes that focus on specific computations can easily be achieved (*e.g.* if we are only interested in the mean value of an attribute).

83. Gupta, Crooks, et al., *op. cit.*
84. Aguilar-Melchor, Barrier, et al., *op. cit.*
85. Benny Chor, Oded Goldreich, et al., « Private information retrieval », *in*: *Proc. of FOCS'95*, 1995, pp. 41–50.
86. Aguilar-Melchor, Barrier, et al., *op. cit.*
87. Cynthia Dwork, « Differential privacy », *in*: *Proc. of ICALP'06*, 2006, pp. 1–12.
88. Dwork and Roth, *op. cit.*
89. Sweeney, *op. cit.*
90. In this manuscript, the word of "anonymization" refers to techniques that make it impossible to identify or single out an individual, in an irreversible way. In this regard, this definition is similar to the one used in the GDPR, and is not to be confused with "pseudonymization". `https://gdpr.eu/recital-26-not-applicable-to-anonymous-data/`

Although this formal model is relatively new (it was first introduced in 2006 [91]), some differentially private processes were actually used in 1970, to estimate the number of abortions in North Carolina [92].

Intuitively, the definition of anonymisation provided by differential privacy states that a function f is anonymised if, for any given dataset D, and any other dataset D' differing from D by one individual, $f(D)$ is close to $f(D')$. In that case, the participation of any individual in the database is not leaked through from f, and therefore, its output can be published without endangering individuals.

More formally, two more details have to be added. First, if f were deterministic (and gave at least two different results), an attacker could deduce information on the database from its result, in an indisputable way. To avoid this issue, differential privacy considers that f is a probabilistic function, meaning that the output $f(D)$ is not deterministic. As a consequence, saying that $f(D)$ has to be close to $f(D')$ means that the output distribution of f on D has to be close to the one on D'. Then, it is also required to add a parameter: ϵ ($\epsilon > 0$) [93]. This parameter sets the proximity allowed between $f(D)$ and $f(D')$: the lower the value of ϵ, the higher the protection, and the least f depends on the dataset. This leads to the formal Definition 1.

Definition 1 (ϵ-differential privacy [94])**.** The randomized function f satisfies ϵ-differential privacy, where $\epsilon > 0$, if:

$$P(f(D) = X) \leq e^{\epsilon} \times P(f(D') = X)$$

for any output $X \in Range(f)$ and any datasets D and D' that differ in at most one individual.

However, transforming a function $\tilde{f}$ into a differentially private function f heavily depends on $\tilde{f}$ itself. Indeed, when $\tilde{f}(D)$ and $\tilde{f}(D')$ differ a lot, transformations to make $\tilde{f}$ differentially private are heavier than when $\tilde{f}(D)$ is already close to $\tilde{f}(D')$. For instance, when counting individuals, $\tilde{f}(D)$ and $\tilde{f}(D')$ differ by at most 1, as D and D' differ by at most one individual. When summing all ages in a database, this difference can be

91. Dwork, *op. cit.*

92. James R Abernathy, Bernard G Greenberg, and Daniel G Horvitz, « Estimates of induced abortion in urban North Carolina », *in*: *Demography* 7.*1* (1970), pp. 19–29.

93. Another parameter δ is sometimes added. As we do not make use of it in this manuscript, we prefer not to introduce it for the sake of simplicity. Interested readers may refer to (Dwork and Roth, *op. cit.*) for more details

94. Dwork, *op. cit.*

as high as the maximum age represented in the database. Therefore, we also introduce the *sensitivity* of a function $\tilde{f}$: $\Delta\tilde{f} = max_{D,D'}|\tilde{f}(D) - \tilde{f}(D')|$, that is useful to enforce differential privacy.

Geometric mechanism In most cases, this definition is enforced by adding random noise to the output, depending on both ϵ and Δf. For instance, the geometric mechanism [95], defined in 2, adds random values drawn from a geometric distribution and parametrized with ϵ and Δf to enforce differential privacy. Note that many other mechanisms, such as the Laplace mechanism [96], exist to allow continuous outputs, or the exponential mechanism [97], useful for categorical attributes as it draws an output among all possible solutions to respect differential privacy.

Definition 2 (Geometric mechanism [98])**.** Let $\mathcal{G}$ denote a random variable following a two-sided geometric distribution, meaning that its probability density function is $g(z, \alpha) = \frac{1-\alpha}{1+\alpha}\alpha^{|z|}$ for $z \in \mathbb{Z}$. Given any function $\mathtt{f} : \mathbb{N}^{|\mathcal{X}|} \to \mathbb{Z}^k$ the Geometric Mechanism is defined as $M_G(x, f(.), \alpha) = \mathtt{f}(x) + (Y_1, \ldots, Y_k)$ where Y_i are independent identically distributed random variables drawn from $\mathcal{G}(e^{-\epsilon/\Delta\mathtt{f}})$, and $\Delta\mathtt{f}$ is its global sensitivity $\Delta\mathtt{f} = \max_{\mathcal{P}_1,\mathcal{P}_2} ||\mathtt{f}(\mathcal{P}_1) - \mathtt{f}(\mathcal{P}_2)||_1$ for all $(\mathcal{P}_1, \mathcal{P}_2)$ pairs of sets of individuals such that $\mathcal{P}_2$ is $\mathcal{P}_1$ with one more individual.

Composition with differential privacy Finally, we present two composition properties of differential privacy in Theorem 1, which allow multiple differentially private processes to be used in a single algorithm, while keeping track of the level of protection guaranteed. We especially highlight that sequential composition ensures that applying an ϵ_1-differentially private process and an ϵ_2-differentially private process results in an $\epsilon_1 + \epsilon_2$-differentially private process. This property explains that the parameter ϵ is also called the *privacy budget*. A common way to design differentially private algorithm is to fix an initial ϵ, and to decompose it along the steps of the algorithm, without exceeding it.

Theorem 1 (Sequential and parallel composability [99])**.** *Let f_i be a set of functions such that each provides ϵ_i-differential privacy. First, the* sequential composability *property of*

95. Arpita Ghosh, Tim Roughgarden, and Mukund Sundararajan, « Universally utility-maximizing privacy mechanisms », *in*: *SIAM Journal on Computing* 41.6 (2012), pp. 1673–1693.
96. Dwork, *op. cit.*
97. Frank McSherry and Kunal Talwar, « Mechanism design via differential privacy », *in*: *Proc. of FOCS'07*, IEEE, 2007, pp. 94–103.
98. Ghosh, Roughgarden, and Sundararajan, *op. cit.*
99. Dwork and Roth, *op. cit.*

differential privacy states that computing all functions on the same dataset results in satisfying $(\sum_i \epsilon_i)$*-differential privacy. Second, the* parallel composability *property states that computing each function on disjoint subsets provides* $\texttt{max}(\epsilon_i)$*-differential privacy.*

Computational differential privacy Computational differential privacy [100] was introduced to compose differentially private processes with cryptography security hypothesis. This relaxation of differential privacy requires the function actually computed to be *computationally indistinguishable* from a pure (information theoretic) ϵ-differentially private function to adversaries whose computational power is limited (this limitation is quantified by a security parameter $\kappa \in \mathbb{N}$). Definition 3 provides a more formal definition for this notion.

Definition 3 (ϵ_κ-SIM-CDP privacy [101] (simplified))**.** The randomized function $\texttt{f}_\kappa$ provides ϵ_κ-SIM-CDP if there exists a function $\texttt{F}_\kappa$ that satisfies ϵ-differential privacy and a negligible function $negl(\cdot)$, such that for every dataset $\mathcal{P}$, every probabilistic polynomial time adversary $\texttt{A}_\kappa$, every auxiliary background knowledge $\zeta_\kappa \in \{0,1\}^*$, it holds that:

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

Infinite divisibility Intuitively, a distribution is said to be infinitely divisible if it can be decomposed as a sum of an arbitrary number of independent identically distributed random variables. This property allows to distribute the generation of the noise over a set of participants. It is valuable in contexts such as ours where no single trusted party, in charge of generating the noise, exists. Definition 4 below formalizes the infinite divisibility property, and Theorem 2 shows that the two-sided geometric distribution is infinitely divisible.

Definition 4 (Infinite divisibility)**.** A probability distribution with characteristic function ψ is infinitely divisible if, for any integer $n \geq 1$, we have $\psi = \phi_n^n$ where ϕ_n is another characteristic function. In other words, a random variable Y with characteristic function ψ has the representation $Y \stackrel{d}{=} \sum_{i=1}^n X_i$ for some independent identically distributed random variables X_i.

100. Ilya Mironov, Omkant Pandey, et al., « Computational Differential Privacy », *in*: *Proc. of CRYPTO'29*, 2009, pp. 126–142.
101. *Ibid.*

Theorem 2 (Two-sided geometric distribution is infinitely divisible)**.** *Let Y follow two-sided geometric distribution with probability density function $d(z,\epsilon) = \frac{1-\epsilon}{1+\epsilon}\epsilon^{|z|}$ for any integer z. Then the distribution of Y is infinitely divisible. Furthermore, for every integer $n \geq 1$, representation of definition 4 holds. Each X_i is distributed as $X_{1n} - X_{2n}$ where X_{1n} and X_{2n} are independent identically distributed random variable with negative binomial distribution, with probability density function $g(k,n) = \binom{k-1+1/n}{k}(1-\alpha)^k * \alpha^{1/n}$.*

To prove this result, we will use a similar property for the geometric distribution.

Theorem 3 (Geometric distribution is infinitely divisible [102])**.** *Let Y have a geometric distribution with probability density function $f(k,\alpha) = (1-\alpha) * \alpha^k$ for $k \in \mathbb{N}$. Then the distribution of Y is infinitely divisible. Furthermore, for every integer $n \geq 1$, representation of 4 holds. Each X_i is distributed as X_n where X_n are independent identically distributed random variable with negative binomial distribution, with probability density function $g(k,n,\alpha) = \binom{k-1+1/n}{k}(1-\alpha)^k\alpha^{1/n}$.*

Proof. Using this theorem, proving that a two-sided geometric distribution with density function $d(z,\alpha) = \frac{1-\alpha}{1+\alpha}\alpha^{|z|}$is equal to the difference between two independent identically distributed geometric distributions with density function $f(k,\alpha) = (1-\alpha)\alpha^k$ is enough to deduce the result. Let X_+ and X_- be two such random variables.

$$P(X_+ - X_- = z) = \begin{cases} \text{if } z \geq 0 \\ \sum_{j=0}^{\infty}((1-\alpha)\alpha^{z+j})((1-\alpha)\alpha^j) \\ \text{if } z < 0 \\ \sum_{j=0}^{\infty}((1-\alpha)\alpha^j)((1-\alpha)\alpha^{-z+j}) \end{cases}$$

$$\begin{aligned} P(X_+ - X_- = z) &= \sum_{j=0}^{\infty}(1-\alpha)^2\alpha^{|z|+2j} \\ &= (1-\alpha)^2\alpha^{|z|}\sum_{j=0}^{\infty}(\alpha^2)^j \\ &= (1-\alpha)^2\alpha^{|z|}\frac{1}{1-\alpha^2} \\ &= \frac{(1-\alpha)^2}{(1-\alpha)(1+\alpha)}\alpha^{|z|} \\ &= \frac{1-\alpha}{1+\alpha}\alpha^{|z|} \\ &= P(Y = z) \end{aligned}$$

for Y a random variable with a two-sided geometric distribution with parameter α.

□

102. Fred W Steutel and Klaas Van Harn, *Infinite divisibility of probability distributions on the real line*, 2003.

2.3.3 Quality measure

We evaluate the quality of the (perturbed) output of the PKD algorithm by measuring the loss of accuracy resulting from its use rather than using non-protected raw profiles. As stated in Definition 5, given a set of tasks T, we compute the average absolute error between (1) the approximate number of workers matching each task $t \in \mathcal{T}$ according to the (perturbed) partitions and counts and (2) the exact non-protected number of matching workers. Note that the error comes from the perturbation used for satisfying differential privacy but also from the inherent approximation due to the use of a coarse grain data structure (partitions and counts) synthesizing raw data.

Definition 5 (Quality)**.** Given a set of worker profiles $\mathcal{P}$, a set of tasks $\mathcal{T}$, and, for each task $t \in \mathcal{T}$, the real number of workers matching with it t_{match} and its approximated value $\widetilde{t_{match}}$ according to the perturbed distribution we provide. We compute the quality of the distribution as

$$Q = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \frac{|t_{match} - \widetilde{t_{match}}|}{t_{match}}$$

We also measure the quality of our assignment by using *precision*, to quantify the number of tasks that are uselessly downloaded by workers, as seen in Definition 6. Note that in our context, all matching tasks will be downloaded, such that a *recall* measure is not relevant as it will always be equal to 1.

Definition 6 (Precision)**.** For a given assignment, we call precision the fraction of downloaded tasks that match with the worker. This precision is computed as:

$$precision = mean_{t \in \mathcal{T}}\left(\frac{|\{w : match(w,t) \wedge download(w,t)\}|}{|\{w : download(w,t)\}|}\right)$$

2.4 The PKD algorithm

Our proposal comes from a rethinking of the centralized version of the *KD-Tree* construction algorithm. A KD-Tree [103] is a well-known data structure designed for partitioning datasets in k-dimensional balanced partitions. It is constructed by recursively dividing the space in two around the median. It is widely used to index data. Moreover, it contains valuable information about the data distribution (it is balanced) without sizing individual

103. Jon Louis Bentley, « Multidimensional binary search trees used for associative searching », *in*: *Communications of the ACM* 18.*9* (1975), pp. 509–517.

data points. In our setting, each worker holds its own, possibly sensitive, profile and no single centralized party is trusted. Centralizing the profiles of workers in order to compute a KD-Tree over them is therefore not possible. A naive approach could make use of an order-preserving encryption scheme [104] (OPE), but these schemes are well-known for their low security level [105] and especially their inherent weaknesses against frequency analysis attacks. We rather favor sound privacy guarantees - without sacrificing efficiency - by approaching the median through the computation of histograms, with a computation distributed between workers and the platform. Similarly to its centralized counterpart, each iteration of our recursive algorithm divides the space of skills in two around the median (of one dimension at a time) given a perturbed histogram representing the distribution of a single skill in the crowd. For simplicity, the current version of the algorithm terminates after a fixed number of splits, but more elaborate termination criteria can be defined.

2.4.1 Computing private medians

From private sum to private histogram We start by explaining how to perform differentially private sums based on noise-shares and additively-homomorphic additions. It allows the platform to get the result of the addition of a single bin over n different workers while satisfying differential privacy. We then show that this function is a sufficient building block for computing perturbed histograms.

Let us consider a fixed $\epsilon > 0$, a maximum size of collusion $\tau \in \mathbb{N}, \tau > 0$, a set of workers $\mathcal{P}$ (of size $|\mathcal{P}|$), and a single bin b_i associated to a range ϕ. Each worker p_i holds a single private local value, *i.e.,* her skill on the dimension that is currently being split. Initially, the bin b_i is set to 0 on all workers. Only the workers whose local values fall within the range of the bin b_i set b_i to 1. Our first goal is to compute the sum of the bins b_i of all workers $p_i \in \mathcal{P}$ such that no set of participant smaller than τ can learn any information that has not been perturbed to satisfy ϵ-differential privacy.

The privacy-preserving sum algorithm, depicted in Fig. 2.2, considers that keys have been generated and distributed to workers, such that $T > \tau$ key-shares are required for decryption (see Section 2.3.2). The algorithm consists in the following steps:

Step 1 (each worker) - Perturbation and encryption First, each worker perturbs

104. Rakesh Agrawal, Jerry Kiernan, et al., « Order-Preserving Encryption for Numeric Data », *in*: *Proc. of SIGMOD'04*, 2004, pp. 563–574.
105. Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill, « Order-Preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions », *in*: *Proc. of CRYPTO'31*, 2011, pp. 578–595.

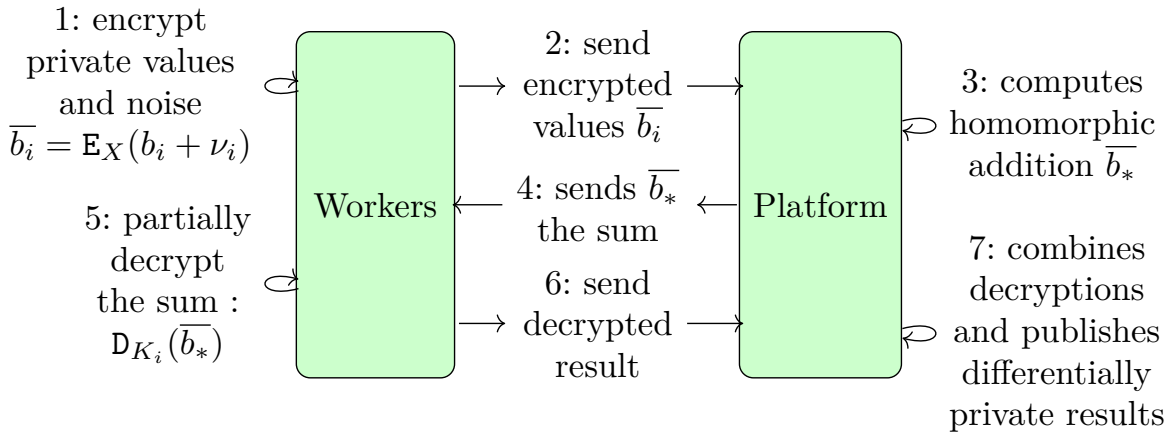


Figure 2.2 – Computing a private sum

its value by adding a noise-share, denoted ν_i, to it. Noise-shares are randomly generated locally such that the sum of $|\mathcal{P}| - \tau$ shares satisfies the two-sided geometric distribution (see Definition 2 for the geometric distribution, and Theorem 2 for its infinite divisibility). Note that noise-shares are overestimated [106] to guarantee that the final result is differentially private even for a group of up to τ workers sharing their partial knowledge of the total noise (their local noise-share). Each worker computes $b_i + \nu_i$ and encrypts it by the additively-homomorphic encryption scheme in order to obtain $\overline{b_i}$: $\overline{b_i} = \mathtt{E}_X(b_i + \nu_i)$.

Step 2 (platform) - Encrypted sum The platform sums up together the encrypted values received : $\overline{b_*} = \sum_{\forall i} \overline{b_i}$ where the sum is based on the additively-homomorphic addition $+_h$.

Step 3 (subset of workers) - Decryption The platform finally sends the encrypted sum $\overline{b_*}$ to at least T distinct workers. Upon reception, each worker partially decrypts it based on her own key-share - $\mathtt{D}_{K_i}(\overline{b_*})$ - and sends it back to the platform. The platform combines the partial decryptions together and obtains $\widetilde{b_*}$, *i.e.,* the differentially private sum of all private bins b_i.

Now, assuming that the histogram format is fixed beforehand - *i.e.,* number l of bins and ranges $(\phi_0, \ldots, \phi_{l-1})$ - it is straightforward to apply the private sum algorithm on each bin for obtaining the perturbed histogram based on which the median can then

106. We require that the sum of $|\mathcal{P}| - \tau$ noise-shares be enough to satisfy differential privacy but we effectively sum $|\mathcal{P}|$ noise-shares. Note that summing more noise-shares than necessary does not jeopardize privacy guarantees.

Algorithm 1: `PrivMed` : Privacy-preserving estimation of the median in the `PKD` algorithm

Data:
$\mathcal{P}$: Set of workers
$\mathcal{D}_{min}, \mathcal{D}_{max}$: Definition domain of the private local value of workers
l: Number of bins
$(\phi_0, \ldots, \phi_{l-1})$: the l ranges of the bins
X : public encryption key (same for all workers).
$\{K_i\}$: private decryption keys (one per worker).
ϵ_m: differential privacy budget for this iteration
τ: maximum size of a coalition ($\tau < |\mathcal{P}|$)
T: number of key-shares required for decryption ($T > \tau$)
Result: $\widetilde{m}$: estimate of the median of the workers' local private values

1 **for** *all workers* $p_i \in \mathcal{P}$ **do**
2 Compute l noise shares $\nu_{i,j} = R_1 - R_2$, where $0 \leq j < l$ and R_1 and R_2 are independent identically distributed random variables with probability density function $g(k) = \binom{k-1+\frac{1}{|\mathcal{P}|-\tau}}{k}(e^{-\epsilon_m})^k(1-e^{-\epsilon_m})^{\frac{1}{|\mathcal{P}|-\tau}}$
3 Set the value of the bin $\overline{b_{i,k}} = \mathtt{E}_X(1+\nu_{i,k})$, where ϕ_k is the histogram range within which the local value of the worker falls.
4 Set the value of the other bins to: $\overline{b_{i,j}} = \mathtt{E}_X(\nu_{i,j})$, $j \neq k$.
5 **Platform:** sum the encrypted bins at the same index received from different workers in order to obtain the encrypted perturbed histogram : $(\overline{b_{*,0}} = \sum_{\forall i} \overline{b_{i,0}}, \ldots, \overline{b_{*,l-1}} = \sum_{\forall i} \overline{b_{i,l-1}})$.
6 **Workers (T distinct workers):** Decrypt partially the encrypted perturbed histogram bin per bin and send the resulting partially decrypted histogram to the platform : $(\mathtt{D}_{K_i}(\overline{b_{*,0}}), \ldots, \mathtt{D}_{K_i}(\overline{b_{*,l-1}}))$.
7 **Platform:** Combine the partial decryptions together to obtain the decryption of the histogram and estimate the median $\widetilde{m}$ according to Equation 2.1.
8 **return** $\widetilde{m}$

be computed. For example, in order to get a histogram representing the distribution of skill values for, *e.g.,* `Python programming`, and assuming a basic histogram format - *e.g.,* skill values normalized in $[0, 1]$, $l = 10$ bins, ranges ($\phi_0 = [0, 0.1[, \ldots, \phi_9 = [0.9, 1]$) - it is sufficient to launch ten private sums to obtain the resulting perturbed 10-bins histogram.

`PrivMed`: privacy-preserving median computation The histogram computed based on the privacy-preserving sum algorithm can be used by the platform to estimate the value of the median around which the split will be performed. When by chance the median falls precisely between two bins (*i.e.,* the sum of the bins on the left is exactly 50% of the total sum, same for the bins on the right) its value is exact. But when the median falls within the range of one bin (*i.e.,* in any other case), an additional hypothesis on the underlying data distribution within the bin must be done in order to be able to estimate the median. For simplicity, we will assume below that the distribution inside each bin is uniform but a more appropriate distribution can be used if known.

$$\widetilde{m} = \mathcal{D}_{min} + \frac{\mathcal{D}_{max} - \mathcal{D}_{min}}{l} \cdot (k + \frac{1}{2} + \frac{\theta_> - \theta_<}{2 \cdot \widetilde{b_{*,k}}}) \tag{2.1}$$

The resulting `PrivMed` algorithm is detailed in Algorithm 1.

Let's consider the histogram obtained by the private sum algorithm. It is made of l bins denoted $(\widetilde{b_{*,0}}, \ldots, \widetilde{b_{*,l-1}})$, and each bin $\widetilde{b_{*,j}}$ is associated to a range ϕ_j. The ranges partition a totally ordered domain ranging from $\mathcal{D}_{min}$ to $\mathcal{D}_{max}$ (*e.g.,* from $\mathcal{D}_{min} = 0$ to $\mathcal{D}_{max} = 1$ on a normalized dimension that has not been split yet). Let ϕ_k denote the range containing the median, θ denote the sum of all the bins - *i.e.,* $\theta = \sum_{i<l} \widetilde{b_{*,i}}$ - and $\theta_<$ (resp. $\theta_>$) the sum of the bins that are strictly before (resp. after) $\widetilde{b_{*,k}}$ - *i.e.,* $\theta_< = \sum_{i<k} \widetilde{b_{*,i}}$ (resp. $\theta_> = \sum_{i>k} \widetilde{b_{*,i}}$). Then, an estimation $\widetilde{m}$ of the median can be computed as follows [107]:

2.4.2 Global execution sequence

Finally, the Privacy-preserving KD-Tree algorithm, `PKD` for short, performs the median estimation described above iteratively until it stops and outputs (1) a partitioning of the space of skills together with (2) the perturbed number of workers within each partition. The perturbed number of workers is computed by using an additional instantiation of

107. Note that in the specific case where the median falls within a bin equal to 0 (*i.e.,* $\widetilde{b_{*,k}} = 0$), then any value within ϕ_k is equivalent.

the private sum algorithm when computing the private medians [108]. We focus below on the setting up of the parameters of the various iterations, and on the possible use of the resulting partitions and counts by the requesters. An overview is given in Algorithm 2.

Main input parameters Intuitively, the privacy budget ϵ input of the PKD algorithm sets an upper bound on the information disclosed along the complete execution of the algorithm - it must be *shared* among the various data structures disclosed. Thus, each iteration inputs a portion of the initial privacy budget such that the sum of all portions remains lower than or equal to ϵ - see the composability property in Theorem 1. Computing a good budget allocation in a tree of histograms is a well-known problem tackled by several related works [109] [110]. In this work, we simply rely on existing privacy budget distribution methods [111], and ϵ is divided as follows. First, ϵ is divided in two parts: one part, denoted ϵ^m, is dedicated to the perturbations of the medians computations (*i.e.,* the bins of the histograms), and the other part, denoted ϵ^c, is dedicated to the perturbation of the number of workers inside each partition. Second, a portion of each of these parts is allocated to each iteration i as follows. For each iteration i such that $0 \leq i \leq h$, where h is the total number of iterations (*i.e.,* the height of the tree in the KD-Tree analogy), the first iteration is h (*i.e.,* the root of the tree) and the last one is 0 (*i.e.,* the leaves of the tree) :

$$\epsilon_i^c = 2^{(h-i)/3}\epsilon^c \frac{\sqrt[3]{2}-1}{2^{(h+1)/3}-1} \tag{2.2}$$

$$\epsilon_i^m = \frac{\epsilon^m}{h} \tag{2.3}$$

Note that similarly to Cormode et al. [112], we set the distribution of ϵ between ϵ^c and ϵ^m as follows: $\epsilon^c = 0.7 \cdot \epsilon$ and $\epsilon^m = 0.3 \cdot \epsilon$. Other distributions could be used.

The PKD algorithm stops after a fixed number of iterations known beforehand. Note that more elaborate termination criteria can be defined (*e.g.,* a threshold on the volume of the subspace or on the count of worker profiles contained). The termination criteria must be chosen carefully because they limit the number of splits of the space of skills

108. Note that the perturbed histograms could have been used for computing these counts but using a dedicated count has been shown to result in an increased precision.
109. Cormode, Procopiuc, et al., *op. cit.*
110. Qardaji, Yang, and Li, *op. cit.*
111. Cormode, Procopiuc, et al., *op. cit.*
112. *Ibid.*

and consequently the number of dimensions of worker profiles that appear in the final subspaces. Ideally, the termination criteria should allow at least one split of all dimensions. However, this may not be possible or suitable in practice because of the limited privacy budget. In this case, similarly to a composite index, a sequence of priority dimensions must be chosen so that the algorithm splits them following the order of the sequence. The dimensions that are not part of the sequence will simply be ignored. Note that the number of dimensions in worker 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 work, we make no assumption on the relative importance of dimensions.

Algorithm 2: The `PKD` algorithm

Data:
$\mathcal{P}$: Set of workers
E the current space of skills of d dimensions
h: height of the KD-Tree
Result: T: A Privacy-preserving KD-tree with approximate counts of workers for each leaf

```
We create T as a single leaf, containing the whole space E and a count of all workers.
while current height is smaller than final height do
    Choose a dimension d (for exemple, next dimension).
    for all leaves of the current tree T do
        Compute m the private median of the space of the leaf, as explained in Section 2.4.1.
        For both subspaces separated by the median, compute a private count as explained in Section 2.4.1.
        Create two leaves, containing the two subspaces and associated counts.
        Replace the current leaf by a node, containing the current space and count, and linking to the two newly created leaves.
    Increment the current height.
Apply post-processing techniques explained in Section 2.4.2.
return Tree T
```

Post-processing the output Considering the successive splits of partitions, we can enhance the quality of the counts of workers by exploiting the natural constraints among the resulting tree of partitions : we know that the number of workers in a parent partition must be equal to the number of workers in the union of its children. *Constrained inference*

techniques have already been studied as a post-processing step to improve the quality of trees of perturbed histograms [113] and then improved and adapted to non-uniform distributions of budget [114]. These constrained inference techniques can be used in our context in a straightforward manner in order to improve the quality of the resulting partitioning. We refer the interested reader to the original papers for details [115] [116].

2.4.3 Complexity analysis

We evaluate here the complexity of the PKD algorithm with respect to the number of encrypted messages computed and sent both to and by the platform. The results are summed up in Table 2.1.

The first step to consider is the number of partitions created in the KD-tree. Seen as an index (with one leaf for each point), the construction of a KD-tree requires $2^{h+1} - 1$ nodes, including $2^h - 1$ internal nodes, where h is the maximum height of the KD-Tree. For each node, an encrypted sum is performed, and for each internal node, a histogram is additionally computed, which require l sums, for a total of $(2^{h+1} - 1) + (l \cdot (2^h - 1))$ sums. These counts all require the participation of every worker: for each count, $|\mathcal{P}|$ encrypted messages are computed and sent.

The platform also sends back encrypted messages for each sum, for decryptions to be performed. For each sum, it sends at least T times the homomorphically computed sum, where T is the threshold number of key-shares required for decryption. For simplicity, we assume that the platform sends the ciphertexts to T workers (these are the only encrypted messages that have to be sent to workers during this protocol). Each contacted worker then answers by an encrypted value (the partial decryption). As a conclusion, the total number of encrypted values sent by the workers to the platform $\mathcal{M}_{\Sigma w}$ is:

$$\mathcal{M}_{\Sigma w} = (|\mathcal{P}| + T) \cdot (l \cdot (2^h - 1) + (2^{h+1} - 1)) \tag{2.4}$$

However, as our computation is distributed among all workers, each worker only sends fewer encrypted messages on average $\overline{\mathcal{M}_w}$.

$$\overline{\mathcal{M}_w} = (1 + \frac{T}{|\mathcal{P}|}) \cdot (l \cdot (2^h - 1) + (2^{h+1} - 1)) \tag{2.5}$$

113. Hay, Rastogi, et al., *op. cit.*
114. Cormode, Procopiuc, et al., *op. cit.*
115. Hay, Rastogi, et al., *op. cit.*
116. Cormode, Procopiuc, et al., *op. cit.*

To the platform	$(\lvert\mathcal{P}\rvert + T) \cdot (l \cdot (2^h - 1) + (2^{h+1} - 1))$
By worker (avg)	$(1 + \frac{T}{\lvert\mathcal{P}\rvert}) \cdot (l \cdot (2^h - 1) + (2^{h+1} - 1))$
By the platform	$T \cdot (l \cdot (2^h - 1) + (2^{h+1} - 1))$

Table 2.1 – Number of encrypted messages sent. $\mathcal{P}$ is the set of workers, T the number of partial keys required for decryption, h the depth of the KD-tree, and l the number of bins per median

Finally, the platform sends $\mathcal{M}_{pf}$ encrypted messages.

$$\mathcal{M}_{pf} = T \cdot (l \cdot (2^h - 1) + (2^{h+1} - 1)) \tag{2.6}$$

2.4.4 Security analysis

The only part of the PKD algorithm that depends on raw data is the private sum. The security analysis thus focuses on proving that a single private sum is secure, and then uses the composability properties (see Theorem 1). Theorem 4 proves that the privacy-preserving sum algorithm is secure. We use this intermediate result in Theorem 5 to prove the security of the complete PKD algorithm.

Theorem 4 (Security of the privacy-preserving sum algorithm)**.** *The privacy-preserving sum algorithm satisfies ϵ_κ-SIM-CDP privacy against coalitions of up to τ participants.*

Proof. (sketch) First, any skill in a profile of a participating worker is first summed up locally with a noise-share, and then encrypted before being sent to the platform. We require the encryption scheme to satisfy semantic security, which means that no computationally-bounded adversary can gain significant knowledge about the data that is encrypted. In other words, the leak due to communicating an encrypted data is negligible. Second, the homomorphically-encrypted additions performed by the platform do not disclose any additional information. Third, the result of the encrypted addition is decrypted by combining $T > \tau$ partial decryptions, where each partial decryption is performed by a distinct worker. The threshold decryption property of the encryption scheme guarantees that no coalition of participants smaller than T can decrypt an encrypted value. The final sum consists in the sum of all private values, to which are added $|\mathcal{P}|$ noise-shares. These shares are computed such that the addition of $|\mathcal{P}| - \tau$ shares is enough to satisfy ϵ-differential privacy. Thanks to the post-processing property of differential privacy, adding noise to a value generated by a differentially-private function does not impact the privacy

level. The addition of τ additional noise-shares consequently allows to resist against coalitions of at most τ participants without thwarting privacy. As a result, since the privacy-preserving sum algorithm is the composition of a semantically secure encryption scheme with an ϵ-differentially private function, it is computationally indistinguishable from a pure differentially private function, and consequently satisfies ϵ_κ-SIM-CDP privacy against coalitions of up to τ participants. □

Theorem 5 (Security of the PKD algorithm)**.** *The PKD algorithm satisfies ϵ_κ-SIM-CDP privacy against coalitions of up to τ participants.*

Proof. (sketch) In the PKD algorithm, any collected information is collected through the PrivMed algorithm based on the privacy-preserving sum algorithm. Since (1) the privacy-preserving sum algorithm satisfies ϵ_κ-SIM-CDP (see Theorem 4) against coalitions of up to τ participants, (2) ϵ_κ-SIM-CDP is composable (see Theorem 1), and (3) the privacy budget distribution is such that the total consumption does not exceed ϵ (see Section 2.4.2), it follows directly that the PKD algorithm satisfies ϵ_κ-SIM-CDP against coalitions of up to τ participants. □

2.5 Privacy-preserving task assignement

Once the design of a task is over, it must be assigned to relevant workers and delivered. Performing that while satisfying differential privacy and at the same time minimizing the number of downloads of the task's content is surprisingly challenging. We already discarded in Section 2.1, for efficiency reasons, the *spamming* approach in which each task is delivered to all workers. More elaborate approaches could try to let the platform filter out irrelevant workers based on the partitioned space output by the PKD algorithm (see the Section 2.4). The partitioned space would be used as an index over workers in addition to its primary task design usage. For example, workers could subscribe to their areas of interest (*e.g.,* by sending an email address to the platform together with the area of interest) and each task would be delivered to a small subset of workers only according to its metadata and to the workers' subscriptions. However, despite their appealing simplicity, these *platform-filtering* approaches disclose unperturbed information about the number of workers per area, which breaks differential privacy, and fixing the leak seems hard (*e.g.,* random additions/deletions of subscriptions, by distributed workers, such that differential privacy is satisfied and the overhead remains low).

We propose an alternative approach, based on Private Information Retrieval (PIR) techniques, to diminish the cost of download on the workers side, while preserving our privacy guarantees.

2.5.1 PIR for crowdsourcing: challenges and naive approaches

The main challenge in applying PIR in our context consists in designing a PIR-library such that no information is disclosed during the retrieval of information, and performance is affordable in real-life scenarios. To help apprehending these two issues, we here present two naive methods that break these conditions and show two extreme uses of PIR: one efficient but unsecure, the other is secure but unefficient.

A first PIR-based approach could consist in performing straightforwardly a PIR protocol between the workers and the platform, while considering the PIR-library as the set of tasks itself. The platform maintains a key-value map that stores the complete set of tasks (the values, bitstrings required to perform the tasks) together with a unique identifier per task (the keys), together with their metadata. The workers download the complete list of tasks identifiers and metadata, select locally the identifiers associated to the metadata that match their profiles, and launch one `PIR-get` function on each of the selected identifiers. However, this naive approach leads to blatant privacy issues through the number of calls to the `PIR-get` function. Indeed, in some cases, the platform could deduce the precise number of workers within a specific subspace of the space of skills: with the knowledge of the number of downloads for each worker [117], it is possible to deduce, for each k, the number of workers downloading k tasks. From that, the platform can deduce that the number of workers located in subspaces where k tasks intersect together, and therefore precise information on their skills. This information, kept undisclosed thanks to the `PKD` algorithm, breaks differential privacy guarantees.

A secure but still naive approach could be to consider the power set of the set of tasks (*i.e.* the set of all possible sets of tasks) as the PIR-library, with padding to all file such that they are all the same size (in bits). After this, a worker chooses the PIR-object corresponding to the set of tasks she intersects with, and uses `PIR-get` on it. Although this method prevents the previously observed breach to appear (all behaviours are identical to the platform since everyone downloads exactly one PIR-object, and all PIR-objects are of

117. Even if the identity of workers is not directly revealed, it is possible to match downloads together to break *unlinkability* and deduce these downloads come from the same individual, for example by using the time of downloads, cookies or other identification techniques

the same size), this method would lead to extremely poor results: as every object of the library is padded to the biggest one, and the biggest set of the super set of tasks is the set of tasks itself, this algorithm is even worse than the spamming approach (everyone downloads at least as much as the sum of all tasks, with computation overheads).

These two naive uses of PIR illustrate two extreme cases: the first one shows that using PIR is not sufficient to ensure privacy, and the second one illustrates that a naive secure use can lead to higher computation costs than the spamming approach. In the following, we introduce a method to regroup tasks together, such that each worker downloads the same number of PIR items (to achieve security), while mitigating performance issues by making these groups of tasks as small as possible.

2.5.2 PIR partitioned packing

The security issue showed in the naive PIR use comes from the fact that the number of downloads directly depends on the profiles of workers. Indeed, as the platform has access to the number of downloads, this link leaks information about workers' skills. In order to break this link, we propose to ensure that each worker downloads the same number of items, whatever their profile is. For simplicity, we fix this number to 1 [118], and call *packing* a PIR library that allows each worker to retrieve all their tasks with only one item, and *bucket* an item of such a library, as seen in Definition 7. We prove in Theorem 6 that any packing fulfills our security model.

We can now formalize the conditions that a PIR library must fulfill in order to both satisfy privacy and allow any worker to download all the tasks she matches with.

Definition 7 (Packing, Bucket)**.** A packing L is a PIR library which fulfills the following conditions:

1. **Security condition** Each worker downloads the same number of buckets. This number is set to 1.
2. **PIR requirement** Each PIR item has the same size in bits (padding is allowed):

$$\forall b_1, b_2 \in L, ||b_1|| = ||b_2||$$

This condition comes from the use of PIR.

118. In general, more files can be downloaded at each worker session, but this does not impact significantly the overall amount of computation and does not impact at all the minimum download size for workers.

3. **Availability condition** For all points in the space of skills, there has to be at least one item containing all tasks matching with this point. In other words, no matter their skills (position in the space), each worker can find a bucket that provides every task they match with.

A **bucket** $b \in L$ is an item of a packing. We note $|b|$ for the number of tasks contained in the bucket b, and $t \in b$ the fact that a task t is included in bucket b. The size in bits of a bucket b is denoted as $||b||$.

Theorem 6. *The use of PIR with libraries which fulfill the packing conditions satisfy* ϵ_κ*-SIM-CDP privacy against coalitions of up to* τ *participants.*

Proof. (sketch) In order to prove the security of packing, we observe that (1) the XPIR protocol has been proven computationally secure [119], such that it satisfies ϵ_κ-SIM-CDP, and (2) the use of packing prevents any sensitive information on workers to leak through the number of downloads. Indeed, Condition 7.1 (security) makes each worker call the PIR-get function only once, such that the behaviours of any two workers are indistinguishable. Therefore, the number of PIR-get calls does not depend on profiles. More precisely, the number of PIR-get can only leak information on the number of workers (which is bigger than or equal to the number of PIR-get calls), which does not depend on their profiles, and is already known by the platform.

□

Before considering how to design an efficient packing scheme, we highlight a few noticeable implications of these conditions. First, due to Condition 7.3 (availability), any worker is matched with at least one bucket. To simplify this model, we propose to focus on a specific kind of packings, that can be seen as a partitioning of the space, where each bucket can be linked to a specific subspace, and where all points are included in at least one of such a subspace. We call *partitioned packing* such a packing (Definition 8).

Definition 8 (Partitioned Packing)**.** A partitioned packing is a packing that fulfills the following conditions:

1. Each bucket is associated with a subspace of the space of skills.
2. A bucket contains exactly the tasks that intersect with the subspace it is associated with (this means that all workers in this subspace will find at least the task they match with in the bucket)

119. Aguilar-Melchor, Barrier, et al., *op. cit.*

3. Subspaces associated with the buckets cover the whole space (from Condition 7.3 (availability)).
4. Subspaces associated with the buckets do not intersect each other

In the following, we will focus on partitioned packing. However, in order not to lose generality, we first prove that these packings do not impact efficiency. Indeed, when it comes to the design of a PIR library, efficiency can be affected by two main issues: the number of items and the size of the largest item (in our case, bucket) impact the communication costs, while the size of the overall library (equal to their product) impacts the computation time on the platform. We show in Theorem 7 that with any packing, we can build a partitioned packing that is equivalent or better.

To prove this theorem, we introduce a specific kind of packing that we call *consistent packing*, defined in Definition 9. Essentially, a consistent packing is a packing where no useless task is added to any bucket: in all buckets b, all tasks match with at least one point (a possible worker profile) which has all her tasks in the bucket b. As a result, a consistent packing avoids cases where tasks are in a bucket, but no worker would download it as the bucket does not match all their needs.

Definition 9 (Consistent packing)**.** A packing P is called consistent if and only if, for all buckets $b \in P$, for all tasks $t \in b$, there exists at least one point w in the subspace of t such that all tasks matching with w are in b:

$$\forall b \in P, \forall t \in b, \exists w, (match(w,t) \wedge \forall t' \in T, match(w,t') \Rightarrow t' \in b)$$

Theorem 7. *For any packing P of tasks, there exists a partitioned packing that either has the same size of buckets, number of buckets, or smaller ones.*

Proof. (sketch) Let P be a packing of tasks that is not partitioned. To prove that a partitioned packing can be created that is more efficient than P, we distinguish two cases. First, we consider that each bucket of P can cover a subspace, containing exactly the tasks that intersect with that subspace (thus fulfilling Conditions 8.1 and 8.2). Then, we prove that any consistent packing (as in Definition 9) fulfills Condition 8.2. After that, we consider the case where P does not fulfill this condition, and create a new, smaller packing P_f from P that is consistent, and therefore fulfills Condition 8.2, and use previous results.

We first consider the case where all buckets of P can cover a subspace while fulfilling Condition 8.2, meaning that each bucket contains exactly the tasks that intersect with the

subspace it covers. In that case, Condition 8.1 is trivially fulfilled. If Condition 8.3 is not fulfilled, this means that there is at least a subspace that is not covered by the packing P. Let w be a point in such a subspace. Since P is a packing, the Condition 7.3 (availability) makes it possible to match any point of the space with at least one bucket. In particular, w can be matched with a bucket b. It is enough to extend the subspace associated with b such that it includes w (note that this extension does not break Condition 8.2). We can proceed that way for any point (or more likely any subspace) that is not covered by a subspace, to associate subspaces to a bucket of P, such that this matching fulfills Conditions 8.1, 8.2 and 8.3. If, in this matching, two subspaces associated with buckets of P intersect, it is trivial to reduce one of them to fulfill Condition 8.4 too. Therefore, if all buckets of P can be matched with a subspace while fulfilling Condition 8.2, the theorem holds, since P is equivalent to a partitioned packing.

It can be noticed that if a packing is consistent (Definition 9), Condition 8.2 is fulfilled. Indeed, if $\forall b \in P, \forall t \in b, \exists w, (match(w,t) \wedge \forall t' \in T, match(w,t') \Rightarrow t' \in b)$ (Definition 9), then, for all b in P, we can take V as the union of the $|b|$ points w described in the equation, one for each task t in b. In that case, Condition 8.2 is fulfilled: for each bucket b and its associated subspace V, all tasks in b intersect with V (by definition, as we took V as the union of one point in each task in b), and b contains all tasks that intersect with V (again, by definition, as each task t' that match with a point of V are in b).

In other words, and using the above case, making a packing consistent is sufficient to create a partitioned packing.

We now consider the case where at least one bucket b of P does not cover a subspace such that Condition 8.2 does not stand. In particular, P is not consistent. This means that there is at least one task $t \in b$ such that for all points w in the subspace of t, there is at least one task t' with which w matches and that is not contained by the bucket b. In other words, no points in t can be matched with the bucket b, as b lacks at least one task for each point of t. As a consequence of the Condition 7.3 (availability), this means that all points in t are matched with another bucket. Therefore, the task t can be removed from bucket b, without breaking the properties of a packing, and without increasing the number of buckets, the minimal size of buckets. We proceed so, by removing all such tasks in all buckets recursively: this trivially ends thanks to the finite number of tasks and buckets. By construction, the final packing P_f is consistent.

Therefore, packing P_f is smaller than P, and fulfills Condition 8.2, and we proved in the first case that a packing that fulfills this condition is equivalent to a partitioned

packing, so P_f is equivalent to a partitioned packing. □

2.5.3 Optimizing the packing

With this secure partitioned packing approach, we can discuss how to optimize the overall complexity. First, it can be noticed that Conditions 7.2 and 7.3 (PIR requirement and availability) set a minimal size of bucket: according to Condition 7.3 (availability), there has to be a bucket containing the largest (in bits) intersection of tasks, and Condition 7.2 (PIR requirement) prevents any bucket from being smaller. Furthermore, this minimum is reachable if we consider a packing that creates a partition for each different intersection of tasks and pad to the largest one. However, by building a different bucket for all the possible intersections of tasks, this packing strategy is likely to lead to a very large number of buckets (*e.g.* if a task's subspace is included in another, this packing leads to two buckets instead of one: one containing both tasks, and the other containing only the largest one as it is a different intersection), while we would like to minimize it (and not only the size of buckets). Therefore, although this packing scheme reaches the minimal size of buckets, we cannot consider it as optimal. However, it illustrates what we call an *acceptable* packing (Definition 10), which will be used to define optimality: a packing in which the size of buckets is minimal.

Definition 10 (Acceptable partitioned packing)**.** Let E be a multi-dimensional space, T a set of tasks, *i.e.* a set of positively weighted hyper-rectangles (the hyper-rectangle is the volume of the task, and the weight is their size in bits, denoted w_t for $t \in T$) in the space E and P a packing of these tasks. We call weight of a packing w_P the size in bits of a bucket in P (due to Condition 7.2 (PIR requirement), this size is unique). We call weight of a point w_p in E the sum of the weights of all tasks in T which match with p.

We call minimal weight m_T of the set of tasks T the maximum weight of a point in E: it is the maximum size a worker could require to download. A partitioning P is called acceptable for T if the size of P is equal to m_T: $m_T = w_P$.

NP-hardness of optimal packing

To define optimality, we take this minimum size of buckets, but also try to minimize the number of buckets (or in an equivalent way, the size of the PIR library), as expressed in Definition 11.

Definition 11 (Optimal partitioned packing)**.** For a set of tasks T, we call optimal packing an acceptable packing that minimizes the number of buckets.

However, we prove in Theorem 8 that determining whether there exists an acceptable packing of size n is NP-hard, and therefore, finding the optimal partitioned packing is also NP-hard.

Theorem 8. *Given a set of tasks T, it is NP-hard in $|T|$ to determine whether there exists an acceptable partitioning of n buckets. We call $\mathcal{P}(T, n)$ this problem.*

Proof. (sketch) To prove that this problem is NP-hard, it is enough to demonstrate that a certain problem $\mathcal{P}^+$ known to be NP-complete can be polynomially reduced to $\mathcal{P}$.

We recall that the Partition Problem is NP-complete [120]. $\mathcal{P}^+(S)$: given a multiset S of N positive integers $n_i, i \in [0, N-1]$, decide whether this multiset can be divided into two submultisets S_1 and S_2 such that the sum of the numbers in S_1 equals the sum of the numbers in S_2, and the union of S_1 and S_2 is included in S.

Let us consider a multiset S and the problem $\mathcal{P}^+(S)$. We assume the existence of a deterministic algorithm A that solves $\mathcal{P}(T, n)$ in a polynomial time in $|T|$. We first distinguish a trivial case where the problem $\mathcal{P}^+(S)$ can be solved in polynomial time. Then, we build an algorithm that uses $\mathcal{P}(T, 3)$ to solve $\mathcal{P}^+(S)$ in polynomial time similarly to the remaining cases, which leads to a contradiction.

We first consider a trivial case: if there exists n_k in S such that $n_k > \sum_{i \in [0,N-1], i \neq k} n_i$, then we return $False$. Deciding whether S falls in that specific case is linear in $|S|$, and so is the computation of the answer. If not, let E be a one dimensional space, with bounds $[0, |S| + 1[$. We build T as a set of $|S| + 1$ tasks ($T = \{t_i, i \in [0, N]\}$), such that no task intersects with each other: therefore, the minimum size m_T of T (from Definition 10) will be the same as the size of the biggest task t in T. The $|S|$ first tasks are all associated with an element of S, while the last one will be used to fix m_T. More precisely, we build T as follows:

— the range of t_i is $[i, i + 1[$

— for $i \neq N$, the weight of t_i is equal to the value of n_i ; $w_{t_N} = \frac{\sum_{i \in [0,N-1]} n_i}{2}$.

Building T and t_{max} is subpolynomial.

By hypothesis, $\forall k, n_k \leq \sum_{i \in [0,N-1], i \neq k} n_i$ (as we dealt with this case previously), and by construction, no hyper-rectangle intersects any other, so the minimal weight is the size

120. Narendra Karmarkar and Richard M Karp, *The Difierencing Method of Set Partitioning*, tech. rep., Technical Report UCB/CSD 82/113, Computer Science Division, University of California, Berkeley, 1982.

of the biggest task, which is the last one: $m_T = max_{t_i}(w_{t_i}) = w_{t_{max}}$. Therefore, if S can be divided into two submultisets S_1 and S_2 of the same size, this size is $\frac{\sum_{i\in[0,N-1]} n_i}{2}$, and $\mathcal{P}(T,3)$ answers $True$.

Reciprocally, if $\mathcal{P}(T,3)$ answers $True$, this means that there exists a packing of size 3 such that no packing is bigger than $m_T = \frac{\sum_{i\in[0,N-1]} n_i}{2}$. In particular, as $w_{t_N} = m_T$, this means that no task is added to the bucket containing it, and that the two remaining buckets contain all tasks $t_i, i \neq N$. If one of these buckets where smaller than m_T, the other would be bigger than m_T (as $m_T = \frac{\sum_{i\in[0,N-1]} n_i}{2}$), and therefore, both buckets weight exactly m_T. Therefore, it is possible to separate S in S_1 and S_2 such that the sum of the numbers in S_1 equals the sum of the S_2 by taking all the elements corresponding to the tasks in the first bucket for S_1, and the elements corresponding to the second bucket for S_2.

Therefore, if we are not in the trivial case treated above, $\mathcal{P}(T,3)$ answers $True$ if and only if $\mathcal{P}^+(S)$ is true in polynomial time. As both deciding whether we are in that trivial case and computing the answer in that trivial case can be computed in polynomial time, an algorithm deciding $\mathcal{P}^+(S)$ in polynomial time can be built. The assumption of $\mathcal{P}(T,n)$ not being NP-hard leads to a polynomial algorithm solving $\mathcal{P}^+$, which is absurd, so $\mathcal{P}(T,n)$ is NP-hard.

□

Static packings

Another point can be highlighted: the difference between what we call *static* packing and *dynamic* partitioning. Indeed, when trying to optimize the use of partitioned buckets, two main approaches can be used: adapt buckets to tasks, or adapt tasks to buckets. In the first case, we consider a fixed set of tasks, and try to build partitions in order to minimize the cost of PIR. On the one hand, this optimization makes it possible to perform the best with any set of tasks. On the other hand, as we consider a fixed set of tasks, we may have to compute a new partitioning when this set evolves (when a task is added or removed, at least when it affects the largest bucket). In the second case however, we consider a fixed partitioning, that is independent from the set of tasks. This method is more likely to be suboptimal, but it avoids heavy computation of optimal packing and allows a greater flexibility in the context of crowdsourcing, by allowing a large variety of choices and policies from the platform, which can even lead to other kinds of optimization. For instance, it allows the platform to manage prices policies (*e.g.* making tasks pay for

each targeted subspace, higher prices for tasks willing to target highly demanded subspaces, etc.), in order to even the load within the whole space, and to reduce the redundancy of tasks within the PIR library (tasks that target more than one partition).

As finding the optimal is NP-hard, we prefer to set aside dynamic packings, as its main asset is the theoretical possibility to reach optimality while remaining unrealistic in a real-life scenario, and focus instead on static packings.

Static packing means that the design of partitions is independent of tasks: the tasks contained within the bucket may change, but not the subspace delimited by the partition. These heuristic packing schemes are not optimal in general but may be affordable in real-life scenarios. We propose to use a simple heuristic static packing scheme, the `PKD PIR Packing`, consisting in using the partitioned space of workers profiles computed primarily for task design purposes: to each leaf partition corresponds a bucket containing all the tasks that have metadata intersecting with it (possibly with padding). The resulting algorithm is presented in Algorithm 3. The accordance of this scheme with the distribution of workers can lead to both useful and efficient buckets (as assessed experimentally, see Section 2.6), and the stability over time of the space partitioning (static approach) makes it easier to design policies to approach optimality through incentives on the task design (rather than through bucket design).

Algorithm 3: `PKD PIR Packing`

Data:
T a Tree computed with the `PKD` algorithm
$\mathcal{T}$ a list of tasks
Result: P: A static partitioned packing depending on workers distribution

```
Create an empty packing P
for all leaves l of the tree T do
    Create a new empty bucket b, assigned to the subvolume of l
    for all tasks t in 𝒯 do
        if t and l intersect then
            Add t to the bucket b
    Add b to P
return Packing P
```

2.6 Experimental validation

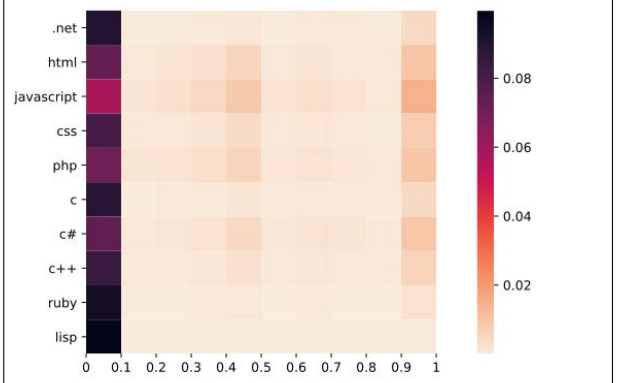
We performed a thorough experimental evaluation of the quality and performances of both the `PKD` algorithm and our `PKD PIR Packing` heuristic (that we abbreviate as *PIR* in the experiments).

2.6.1 Datasets

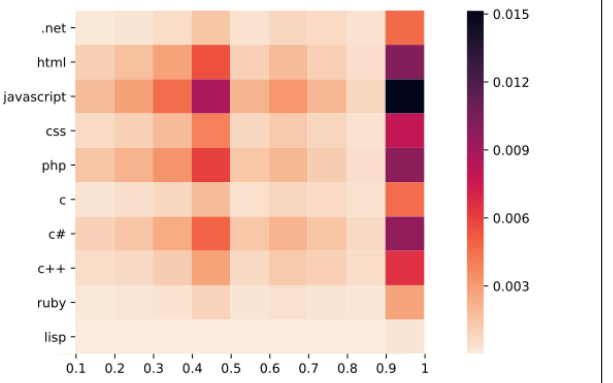
In this section, we introduce the datasets and data generators that are used in our experiments.

Realistic Dataset To the best of our knowledge there does not exist any reference dataset of worker profiles that we could use for our experiments. This led us to building our own dataset from public open data. The *StackExchange*[121] data dumps are well-known in works related to experts finding. We decided to use them as well in order to perform experiments on realistic skills profiles. We computed profiles by extracting skills from users' posts and votes. In *StackExchange*, users submit posts (questions or answers) that are tagged with descriptive keywords (*e.g.,* "python programming") and vote positively (resp. negatively) for the good (resp. bad) answers. We consider then that each user is a worker, that each tag is a skill, and that the level of expertise of a given user on a given skill is reflected on the votes. We favored a simple approach for computing the expertise of users. First, for each post, we compute a *popularity ratio* as follows: $r = \texttt{upvotes}/(\texttt{upvotes} + \texttt{downvotes})$, where `upvotes` is the number of positive votes of the post and `downvotes` is the number of negative votes. Second, for each user p_i, for each tag j, the aggregate level of expertise $p_i[j]$ is simply the average popularity ratio of the posts from p tagged by j. Note that more elaborate approaches can be used[122]. Finally, we removed the workers that do not have any skill level higher than 0. We applied this method on three *StackExchange* datasets: `stackoverflow.com-Posts.7z`, `stackoverflow.com-Tags.7z`, and `stackoverflow.com-Votes.7z` which resulted in 1.3M worker profiles[123]. Figure 2.3

121. *StackExchange* is a set of online forums where users post questions and answers, and vote for good answers `https://archive.org/download/stackexchange`.
122. Ivan Srba and Maria Bielikova, « A comprehensive survey and classification of approaches for community question answering », *in*: *ACM TWEB* 10.*3* (2016), p. 18.
123. The scripts for generating our dataset are available online: `https://gitlab.inria.fr/crowdguard-public/data/workers-stackoverflow`



a) On the x-axis: Skill level. On the y-axis: Skill. On the heatmap: frequency of the given skill-level on the given skill within workers - *including* the workers having a skill level at 0.



b) On the x-axis: Skill level. On the y-axis: Skill. On the heatmap: frequency of the given skill-level on the given skill within workers - *excluding* the workers having a skill level at 0.

Figure 2.3 – Frequencies of ten common skills within the `STACK` dataset.

(a) shows for ten common skills [124] and for the possible levels divided in ten ranges (*i.e.,* $[0.0, 0.1[$, $[0.1, 0.2[$, $\ldots$, $[0.9, 1]$) their corresponding frequencies. It shows essentially that whatever the skill considered, most workers have a skill level at 0. The rest of the distribution is not visible on this graph so we show in Figure 2.3 (b) the same graph but *excluding*, for each tag, the workers having a skill level at 0.

Data Generators We performed our experiments over both synthetic and realistic data. Our two synthetic generators are specifically dedicated to evaluating the `PKD` algorithm with two different kinds of assumptions. First, our `UNIF` synthetic data generator draws skills uniformly at random between 0 and 1 (included) (1) for each dimension of a worker's profile and (2) for each dimension of a task (more precisely, a min value and a max value per dimension). Second, our `ONESPE` generator considers that workers are skilled over a single dimension and that tasks look for workers over a single dimension. The specialty of each worker is chosen uniformly at random, and its value is drawn uniformly at random between 0.5 and 1. The other skills are drawn uniformly at random between 0 and 0.5. Similarly to workers, the specialty looked for by a task is chosen uniformly at random as well, its min value is chosen uniformly at random between 0.5 and 1 and its max value is

124. The ten common skills considered are the following: `.net`, `html`, `javascript`, `css`, `php`, `c`, `c#`, `c++`, `ruby`, `lisp`.

set to 1. The min values of the other dimensions of a task are 0, and their max values are chosen uniformly at random between 0 and 0.5. Although this second heuristic is obviously not perfect, it seems far more realistic than the previous one. For the two task generation heuristics, we require that all tasks must contain at least one worker so that the quality can be correctly computed.

Finally, our realistic data generator, called `STACK`, consists in sampling randomly workers (by default with a uniform probability) from the `STACK` dataset. For our experiments, we generated through `STACK` workers uniformly at random and performed the `ONESPE` task generation strategy described above.

2.6.2 `PKD` algorithm

Quality of the `PKD` algorithm For our experiments, we implemented the `PKD` algorithm in Python 3 and run our experimental evaluation on commodity hardware (Linux OS, 8GB RAM, dual core 2.4GHz). In our experiments, each measure is performed 5 times (the bars in our graphs stand for confidence interval), $1k$ tasks, 10 dimensions, and $\tau = 1$.

In Fig. 2.4 (a), we fix the privacy budget to $\epsilon = 0.1$, the number of bins to 10, and the number of workers to $10k$, and we study the impact of the depth of the tree on the quality. `UNIF` achieves the lowest error, as long as the tree is not too deep. This can be explained by the uniform distribution used in the generation method, which matches the uniform assumption within leaves in the tree. When the depth (and the number of leaves) grows, this assumption matters less and less. `ONESPE` is more challenging for the `PKD` algorithm because it is biased towards a single skill. It achieves a higher error but seems to benefit from deeper trees. Indeed, deep trees may be helpful in spotting more accurately the specialized worker targeted. The results for `STACK` are very similar. For all of these distributions, we can see that having a tree deeper than the number of dimensions leads to a significant loss in quality.

In Fig. 2.4 (b), we analyze the variations of quality according to the value of ϵ, with 10 bins, a depth of 10, and $10k$ workers. In this case, the relative error seems to converge to a non-zero minimum when ϵ grows, probably due to inherent limits of KD-Tree's precision for tasks.

In Fig. 2.4 (c), we fix the privacy budget to $\epsilon = 0.1$, the depth of the tree to 10 and $10k$ workers. We can see the impact of the number of bins for each histogram used to compute a secure median. This value does not greatly impact the relative error for the `UNIF` and `STACK` models, although we can see that performing with 1 bin seems to give

slightly less interesting results, as it looses its adaptability toward distributions. For the `ONESPE` model, having only 1 bin gives better results: indeed, the uniformity assumption within the bin implies that all dimensions are cut at 0.5, which is also by construction the most important value to classify workers generated with this procedure.

In Fig. 2.4 (d), we compare the quality according to the number of workers with $\epsilon = 0.1$, 10 bins and a depths of 10. As the ϵ budget is the same, the noise is independent from this number, and thus, the quality increases with the number of workers.

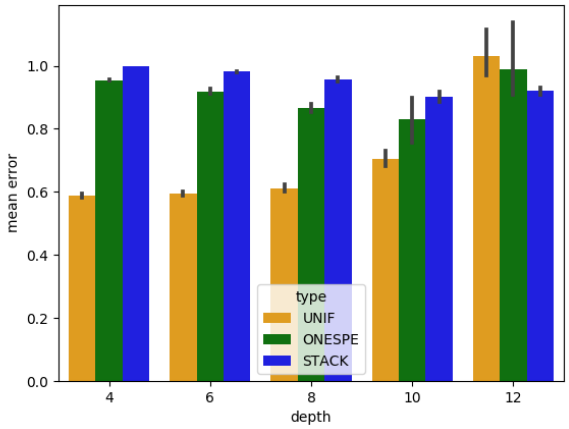
We can notice that our results for the relative error are quite close to the state of the art results, such as the experiments from Cormode et al. [125], which are performed on 2-dimensional spaces only, with strong restrictions on the shapes of queries (tasks in our context) and in a centralized context.

Computation time of the `PKD` algorithm Our performance experiments were performed on a laptop running Linux OS, equipped with $16GB$ of RAM and an Intel Core $i7-7600U$ processor. We measured the average computation time across 100 experiments of each of the atomic operations used in the `PKD` algorithm: encryption, partial decryption, and encrypted addition. The results are summed up in Fig. 2.5, with keys of size 2048 bits, using the Paillier implementation of the University of Texas at Dallas [126]. We use our cost analysis together with these atomic measures for estimating the global cost of the `PKD` algorithm over large populations of workers (see Equation 2.4, Equation 2.5, and Equation 2.6 in Section 2.4.3).

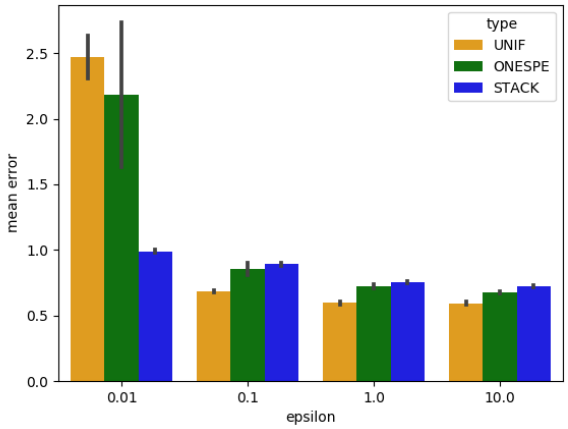
We can observe that the slowest operation is by far the generation of the keys. However, since this operation is performed only once, the cost of less than 1000 seconds (about 17 minutes) for $10k$ workers is very reasonable: this operation can be performed as soon as there are enough subscriptions, and the keys may be distributed whenever the workers connect. The other operations are faster individually, but they are also performed more often. For $10k$ workers, 10 workers required for decryption, a depth of the KD-Tree of 10 and 10 bins, we can observe that: each worker will spend less than 10 seconds performing encryptions, the platform will spend less than 1000 seconds performing encrypted additions, the average worker will spend less than 1 second performing decryptions, and the platform will spend less than 3000 seconds performing decryptions.

Overall, these costs are quite light on the worker side: less than 20 seconds with

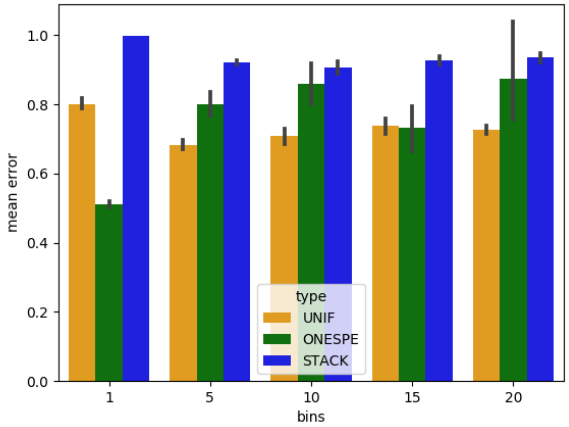
125. Cormode, Procopiuc, et al., *op. cit.*
126. `http://cs.utdallas.edu/dspl/cgi-bin/pailliertoolbox/index.php?go=download`



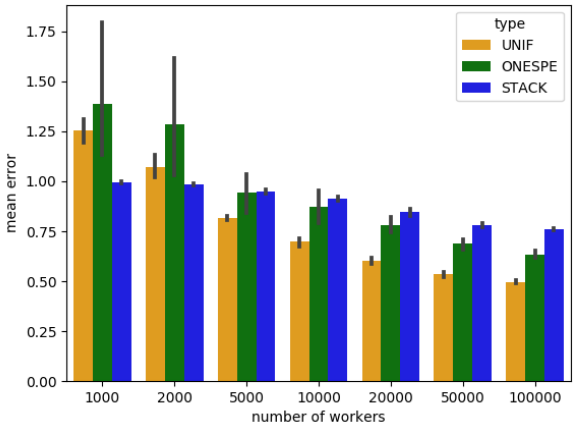
a) Variations according to the depth of the tree.
10 dimensions, $10k$ workers, $1k$ tasks, $\tau = 1$, $\epsilon = 0.1$, 10 bins



b) Variations according to ϵ privacy budget.
10 dimensions, $10k$ workers, $1k$ tasks, $\tau = 1$, 10 bins, $depth = 10$



c) Variations according to the number of bins.
10 dimensions, $10k$ workers, $1k$ tasks, $\tau = 1$, $\epsilon = 0.1$, $depth = 10$



d) Variations according to the number of workers.
10 dimensions, $1k$ tasks, $\tau = 1$, $\epsilon = 0.1$, 10 bins, $depth = 10$

Figure 2.4 – Quality (see Definition 5, the lower the better)

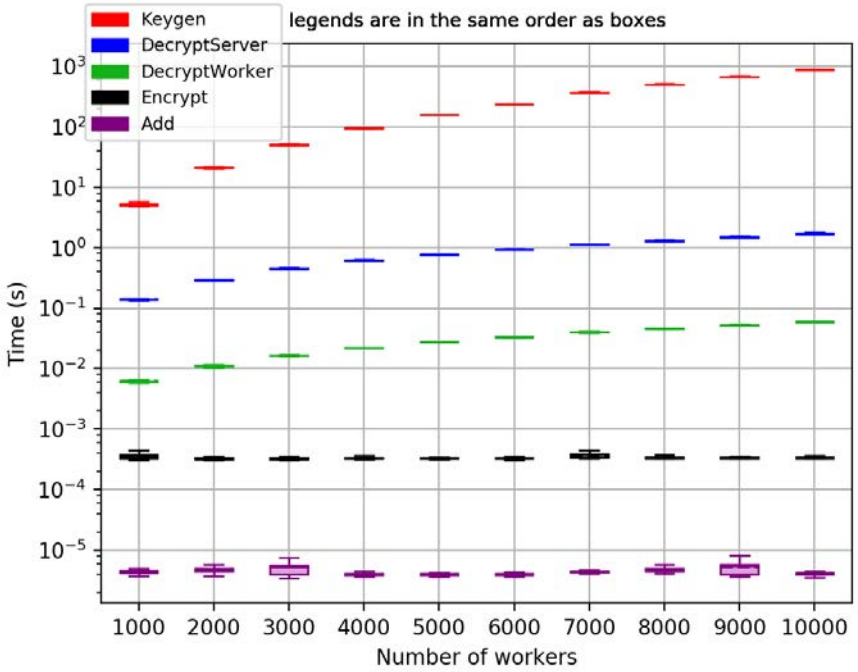
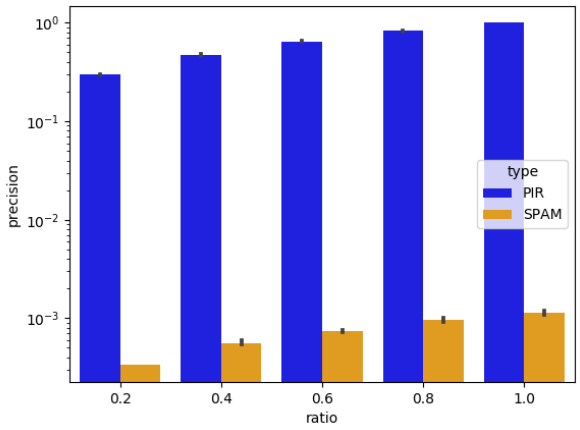


Figure 2.5 – Computation time of homomorphically encrypted operations

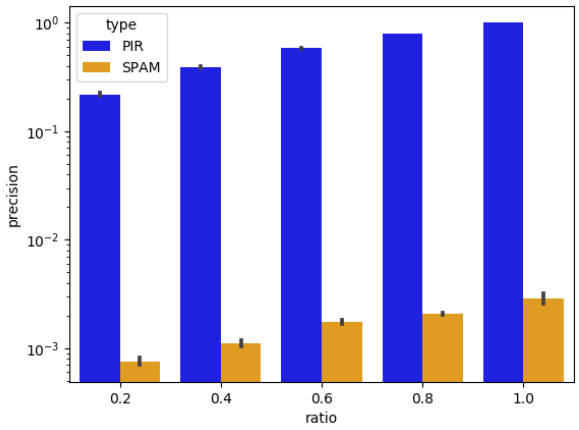
commodity hardware. On the server side, the computation is more expensive (about one hour), but we could expect a server to run on a machine more powerful than the one we used in our experiments. Additionally, it is worth to note that: (1) the perturbed skills distribution is computed only once for a given population of workers and then used repeatedly, and (2) we do not have any real time constraints so that the `PKD` algorithm can run in background in an opportunistic manner.

2.6.3 Assignment using packing

Quality of our packing We here propose to evaluate the quality of our partitioned packing approach. Our experiments are performed with the same settings as those used to measure the quality of the `PKD` algorithm (see Section 2.6.2). To do so, we propose two main metrics. First, we measure the mean precision for tasks, as defined in Definition 6. Although this measure is useful to understand the overall improvement of our approach, it does not take into account the fact that downloads caused by PIR scale with the largest item. Therefore, we introduce a second measure, the mean number of tasks that a worker would download. This value, that we call *maximum tasks*, is computed as the maximum number of tasks that a leaf of the KD-tree intersects with: indeed, due to Condition 7.2



a) Precision in log scale, according to the ratio of leaf taken by task for the UNIF model.
10 dimensions, $10k$ workers, $1k$ tasks, $\tau = 1$, $\epsilon = 0.1$, 10 bins, $depth = 10$



b) Precision in log scale, according to the ratio of leaf taken by task for the ONESPE model.
10 dimensions, $10k$ workers, $1k$ tasks, $\tau = 1$, $\epsilon = 0.1$, 10 bins, $depth = 10$

Figure 2.6 – Precision (the higher the better)

(PIR requirement), all workers will download as many data as contained in the biggest bucket.

In the task generation methods introduced previously, tasks are built independently from the KD-tree itself. This independence was logical to measure the quality of the PKD algorithm. However, this very independence leads to poor results when it comes to building efficient packing on top of a KD-tree: as tasks are independent from the KD-tree, they have little restriction on how small they are (meaning that few workers will match with them, although all workers in leaf that intersect with it will download it), or on how many leaves they intersect with, leading to low precision, and high size of buckets.

Therefore, we introduce a new method to build tasks: SUBVOLUME. With this method, we build tasks as *subleaves*, meaning that all tasks are strictly included within *one* leaf of the KD-tree. Furthermore, we also enforce the size of the task as a parameter, such that the volume of the task is equal to a given ratio of the task. More precisely, for a ratio $r \in [0, 1]$, a space E of d dimensions and a picked leaf l, the interval of a task in a given dimension d_i $l_{d_i} \times r^{1/d}$, where l_{d_i} is the interval of the leaf in dimension d_i. The SUBVOLUME model of tasks can easily be introduced by economic incentives from the platform, such as having requesters pay for each targeted leaf, which is likely to induce a maximization of the volume taken, and a reduction of the tasks that intersect with more than one leaf. Note

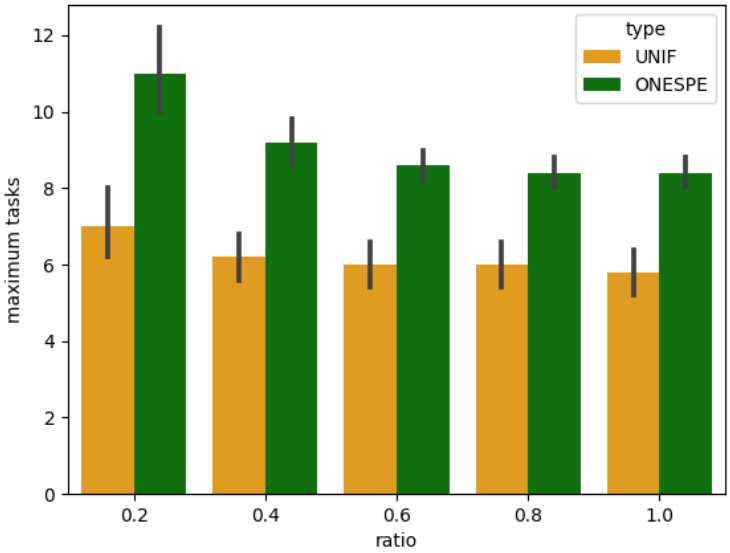


Figure 2.7 – Number of tasks downloaded according to the ratio of leaf taken by task for the packing approach. 10 dimensions, $10k$ workers, $1k$ tasks, $\tau = 1$, $\epsilon = 0.1$, 10 bins, $depth = 10$

that we do not perform experiments with this generation of tasks on the `Stack` dataset, as most workers have their skills set to either 0 or 1, which leads to very unreliable results as tasks almost never encompass either of these values.

The comparison between the `PKD PIR Packing` heuristic and the spamming approach using this new method to generate tasks, presented in Figure 2.6, shows that our approach improves precision by at least two orders of magnitude. Also, note that for $r = 1$, the precision is equal to 1 in the PIR approach. This result comes from the fact that, with $r = 1$, all workers within a leaf are targeted by all tasks that intersect with that leaf, meaning that they do not download irrelevant tasks.

The maximum number of tasks connected to a leaf, showed in Figure 2.7, shows that the cost of download is also significantly improved (these values are to be compared to 1000, the total number of tasks that are downloaded with the spamming approach) also shows great improvement (around 2 orders of magnitude), as tasks are more evenly spread within the leaves (there are $2^{10} = 1024$ leaves for a depth 10 of the tree, which can explain this improvement).

Cost of the PIR protocol We evaluate our heuristic by first measuring experimentally the efficiency of the PIR protocol, and then proposing a cost model that builds on these measures.

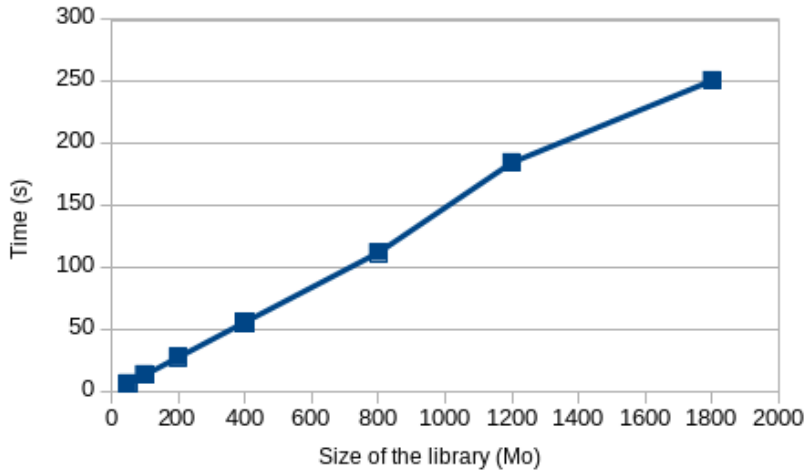


Figure 2.8 – Computation time of the retrieval of an item according to the PIR library's size

We study the impact of the number of files and of the size of files on the computation time. In the experiments, we used a computer with $8GB$ of RAM, and a Ryzen 5 1700 processor, using the implementation of Aguilar-Melchor et al. [127] [128].

As we can see in Figure 2.8 with keys of size 1024 bits, computation time is proportional to the overall size of the PIR library (the coefficient of determination gives $r^2 = 0.9963$), and that it grows at $0.14s/MB$ for a given request, as long as the library can be stored in RAM.

We now evaluate the maximum number of tasks n_{max} that our system can take into account, according to two parameters: the time t that workers accept to wait before the download begins, and the size s that workers accept to download. As n_{max} does not solely depend on t and s, we introduce a few other notations:

— f is the expansion factor of the encryption scheme.
— $|task|$ the mean size of a task.
— k the proportion of tasks that are in the biggest leaf of the KD-tree (for instance, $k = 0.1$ means that the biggest leaf contains one tenth of all tasks)
— $depth$, the depth of the KD-tree (that is linked with the number of buckets)

In the spamming approach, the maximum number of tasks that can be managed by

127. Aguilar-Melchor, Barrier, et al., *op. cit.*
128. `https://github.com/XPIR-team/XPIR`

our system is independent from t and can be simply computed as:

$$n_{max,SPAM} = \frac{s}{|task|}$$

For the `PKD PIR Packing` heuristic, both s and t lead to a limitation on $n_{max,PIR}$. We first consider the limit on the computation time t: according to our results in Figure 2.8, the PIR library cannot be bigger than $\frac{t}{0.14}$, and the size of a bucket, can be computed as $k \times |task| \times n_{max,PIR}$ (by definition of k, as all buckets weight as much as the biggest one). As the library can be computed as the product of the number of buckets and their size, this leads us to $2^{depth} \times k \times |task| \times n_{max,PIR} \leq \frac{t}{0.14}$, or equivalently $n_{max,PIR} \leq \frac{t}{0.14 \times 2^{depth} \times k \times |task|}$. We now consider the limit s on the size of download. For each worker, the size of a download will be the same, computed as the product of the expansion factor and the size of a bucket: $f \times k \times n_{max,PIR} \times |task| \leq s$. This inequality leads to $n_{max,PIR} \leq \frac{s}{f \times |task| \times k}$. By combining these two inequalities, $n_{max,PIR}$ takes its maximum value when

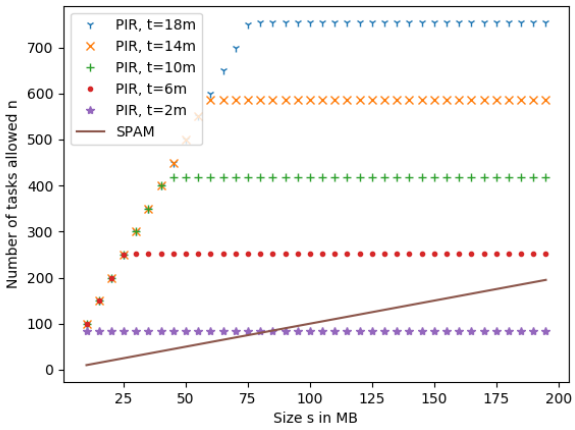
$$n_{max,PIR} = min(\frac{s}{f \times |task| \times k}, \frac{t}{2^{depth} \times 0.14 \times |task| \times k})$$

In Figure 2.9, we compare the number of tasks that a crowdsourcing platform can manage with different values of t and s, using either the spamming approach or our `PKD PIR Packing` heuristic. For the sake of simplicity, we consider that the expansion factor f is 10, although smaller values are reachable with XPIR protocol [129]. This factor will impact the amount of tasks that a worker can download. We take $d = 10$ similarly to our previous experiments. We consider a mean size of task $|task| = 1MB$. It can be noticed that $|task|$ has no impact on the comparison ($\frac{max_{n,PIR}}{max_{n,SPAM}}$ does not depend on $|task|$).

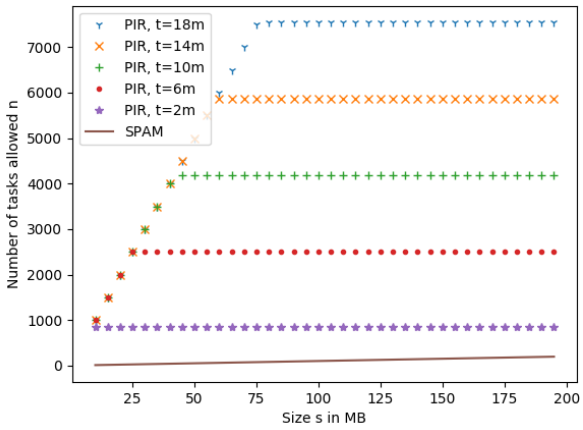
For k, we consider two possible values: $k = 0.01$, as suggested by the experiments in Figure 2.7, and $k = \frac{1}{2^{10}}$, which represents the optimal case, where tasks are perfectly spread among buckets (for instance, due to strong incentives from the platform).

In these experiments, we can notice that our approach depends on both the computation time allowed and the size of the number of task in the largest bucket. In a real-life scenario, platforms would benefit from enforcing incentives to even the load between buckets. However, if workers are willing to limit their download to less than $100MB$, the `PKD PIR Packing` heuristic outperforms the spamming approach as long as users are willing to limit their download even with relatively short computation times (less than 10 minutes) by up

129. Aguilar-Melchor, Barrier, et al., *op. cit.*



a) Number of tasks n manageable by our system according to the size s a worker accepts to download.
$k = 0.01$, $|task| = 1MB$, $f = 10$,
$depth = 10$



b) Number of tasks n manageable by our system according to the size s a worker accepts to download.
$k = 0.001$, $|task| = 1MB$, $f = 10$,
$depth = 10$

Figure 2.9 – Precision (the higher the better) ; curves are in the same order as the captions

to several orders of magnitude. Our method is especially interesting in settings where the bandwidth is low (*e.g.* with mobile devices), with low values of s. On the opposite, it is interesting to highlight that high computation times are not necessarily prohibitive: as the computation is performed by the platform, a worker could very well ask for a bucket of tasks and download it later on when it is ready.

2.7 Discussion

In this section, we propose a discussion on questions raised by our work that are not our primary focus. More precisely, we elaborate our views on updates that our system may or may not allow (both for the `PKD` algorithm and the `PKD PIR Packing` heuristic), with some advantages and drawbacks.

2.7.1 Updating tasks and PIR libraries

In this work, we dealt with the download of tasks as a *one-shot* download, meaning that a worker will download tasks once and for all. However, in a real-life scenario tasks are likely to evolve (*e.g.* new tasks will be added and old tasks will be outdated), and

workers are equally likely to update their tasks. Without further improvement, our design would require each worker to download a whole packing for each update of the available tasks. However, more elaborate approaches are possible. Although it is not our focus to develop them exhaustively, we propose a few tracks that are likely to diminish the costs greatly.

For that purpose, we propose to divide time into fixed duration *periods* (*e.g.* a day, a week, etc.) and to additionally take into account the period at which a task is issued in order to pack it. We give below two options for allowing updates. These schemes result into an increase of the memory cost on the server side, but alleviates the overall computation required.

Packing by Period

A simple scheme that allows easier updates while reducing the size of single PIR request consists in designing packing not only according to a specific partitioning but also according to time periods. The platform builds one PIR library per period, *i.e.,* considering only the tasks received during that period. [130]. Workers simply need to perform PIR requests over the missing period(s) (one request per missing period). As a result, the `PIR-get` function is executed on the library of the requested period, which is smaller than or equal to the initial library.

However, this scheme may result in high costs if the distribution of tasks is skewed. For instance, let's consider two time periods p_1 and p_2, two subspaces of the space of skills s_1 and s_2, and three tasks t_1, t_2 and t_3 such that t_1 and t_2 appear only in p_1 and s_1, while t_3 appears only in p_2 and s_2. In that case, all workers will download first the PIR item for period p_1, which is the same size as $w_{t_1} + w_{t_2}$ (due to padding for workers not in p_1) and then a second PIR item for p_2, of size w_{t_3}. Without that period strategy, a worker who performs regular updates would have downloaded tasks t_1 and t_2 (or equivalent size) twice due to the update, and t_3 once, but a worker who would not have performed the intermediary download would have downloaded $max(w_{t_3}, w_{t_1} + w_{t_2})$. Therefore workers who update frequently would benefit from this strategy, while workers who do not would have worse results.

130. In this kind of methods, a task can be either maintained into its starting period up till its lifespan, or one can consider keeping up a limited number of periods (*e.g.* all daily periods for the current month) and re-adding tasks on new periods packing each time they are deleted (*e.g.* for tasks that are meant to be longer than a month). More elaborate or intermediate methods are also possible, but we will not explore this compromise in this work.

Personalized Packing by Period

In order to tackle the previously mentioned issue caused by skewed distribution of tasks, and to optimize the size of the downloaded bucket for any frequency of downloads, we propose to adapt the packing to the workers' frequency of downloads.

Indeed, we observe that it is enough to perform as many packings as there are possible time-lapses for workers, *e.g.,* one packing for the last period, one packing for the last two periods, one packing for the last three periods, *etc.*. As a result, each `PIR-get` request is associated with a time-lapse in order to let the PIR server compute the buckets to be downloaded (or use pre-computed buckets). With this method, we can get the best of both worlds with the previous example: someone who downloads frequently will only have small updates, while someone who does not will not suffer from overcosts.

The main (and limited) drawback of this method is that the platform will have to store multiple PIR-libraries, which increases the storage required.

Security of Packing by Period

In both of the above schemes, we consider multiple downloads from workers. Even worse, in the second case the number of downloads may vary depending on workers' habits. If the above proposition were to be used, more accurate proofs of security would have to be done. Although it is not our focus to propose them in this chapter, we provide here some intuitions on their requirements. In the first case, the number of downloads is the same for all workers, and would therefore not lead to great modifications of our proof. In the second case however, the number of downloads depends on the frequency of downloads of workers. In order not to reveal information about worker's profiles, a new hypothesis is likely to be required, that states or implies that the frequency of downloads of workers is independent from their profiles.

2.7.2 Updating `PKD`

The `PKD` algorithm is not meant to allow users to update their profiles, as they would have to communicate information to do it, and this would either break our security policy, or exceed the ϵ privacy budget. However, departures or arrivals are not inherently forbidden by our security policy. A simple and naive way to upgrade the `PKD` algorithm to take new arrivals into account is to create multiple KD-trees, and to combine them. For instance, one could imagine using the `PKD` algorithm on every new k arrivals (*e.g.* $k = 1000$ or

$k = 10000$). The estimation of the number of workers within a subspace would be the sum of the estimations for each KD-tree, and a new PIR library could be built for each of these KD-trees. For retrieval of workers, as it is impossible to know where the worker was, the most naive way to proceed is to retrieve a given value to each leaf of the approximated KD-tree, for instance $\frac{n_{leaf}}{n_{tree}}$, where n_{leaf} is the approximated number of workers in the leaf, and n_{tree} the total number of workers. Once again, more elaborate methods are possible, but stand out of our focus.

2.8 Demonstration

In this section, we present the demonstration that is based on this work. This demonstration focuses on secondary usages and the task-tuning in particular, and does not implement the assignment part.

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.

2.8.1 Platform

Figure 2.10 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 be handled by Django without causing a loss of the user experience (i.e., tasks that last more than a few seconds

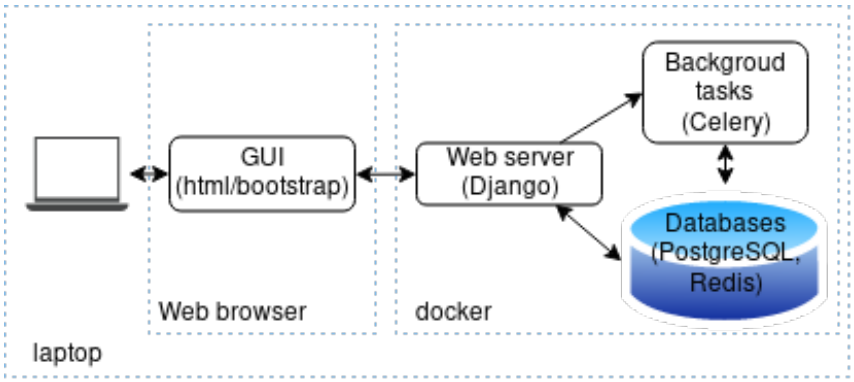


Figure 2.10 – The demonstration runs on a single laptop executing the demonstration platform: the web server (Python Django), the background tasks handler (Celery) and the databases (PostgreSQL and Redis). The demonstration is accessible through a web interface (e.g., on the browser of the demonstration laptop).

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 [131]. 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 [132] and can be executed on a laptop where docker is installed (no configuration is required).

2.8.2 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).

131. In a real-life scenario, encrypted operations would be performed in parallel by workers and the platform itself, which greatly reduces the costs
132. `https://gitlab.inria.fr/crowdguard-public/implems/pkd-demo`

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 [133]. 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.

2.8.3 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 **pkdtr** for details), and one population is computed from the public *Stackoverflow* dataset [134]. 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 2.11.

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

Figure 2.11 – Format of a worker dataset and illustration on a single worker.

2.8.4 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

133. It also impacts the overall computation time and quality of the estimation

134. 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).



Figure 2.12 – 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.

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 2.12 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.

2.9 Conclusion

We have presented a privacy-preserving approach dedicated to enabling various usages of worker profiles by the platform or by requesters, including in particular the design of tasks according to the actual distribution of skills of a population of workers. We have proposed the `PKD` algorithm, an algorithm resulting from rethinking the *KD-tree* construction algorithm and combining additively-homomorphic encryption with differentially-private perturbation. No trusted centralized platform is needed: the `PKD` algorithm is distributed between workers and the platform. We have provided formal security proofs and complexity analysis, and an extensive experimental evaluation over synthetic and realistic data that shows that the `PKD` algorithm can be used even with a low privacy budget and with a reasonable number of skills. Exciting future works especially include considering stronger attack models (*e.g.,* a malicious platform), protecting the tasks in addition to worker profiles, and guaranteeing the integrity of worker profiles.

Chapter 3

SEPAR: A PRIVACY-PRESERVING APPROACH TO REGULATE CROWDSOURCING

3.1 Introduction

The rise of the *platform economy* [1] [2] is reshaping work all around the world. By providing requesters (resp. workers) 24/7 access to a worldwide workforce (resp. worlwide task market), crowdsourcing platforms have grown in numbers, diversity, and adoption. Today, workers in crowdsourcing come from countries spread all over the world, and work on several, possibly competing, platforms [3]. The use of crowdsourcing platforms is expected to continue growing [4], and in fact they are envisioned as key technological components of the future of work [5].

Crowdsourcing platforms, however, challenge national boundaries, weaken the formal relationships between workers and requesters, and are often not considered as legal employers. Guaranteeing the compliance of crowdsourcing platforms with national or regional labour laws is hard [6] [7] despite the stringent need for regulating work. For example, the preamble of the 1919 constitution of the International Labour Organization [8], written

1. Julie E Cohen, « Law for the platform economy », *in*: *UC Davis Law Review, Forthcoming* 51 (2017), p. 133.
2. Martin Kenney and John Zysman, « The rise of the platform economy », *in*: *Issues in science and technology* 32.*3* (2016), p. 61.
3. Berg, Furrer, et al., *op. cit.*
4. Global Commission on the Future of Work, *Work for a brighter future*, tech. rep., International Labour Organization, 2019.
5. participants, *op. cit.*
6. See, e.g., the `Otey V Crowdflower` class action against a famous microtask platform for *"substandard wages and oppressive working hours"* (`https://casetext.com/case/otey-v-crowdflower-1`).
7. Future of Work, *op. cit.*
8. Commission on International Labour Legislation, *Constitution of the International Labour Organi-*

in the ruins of World War I, states that: *"Whereas universal and lasting peace can be established only if it is based upon social justice; (...) an improvement of those conditions is urgently required; as, for example, by the regulation of the hours of work, including the establishment of a maximum working day and week, the regulation of the labour supply (...)".* The global regulation of the work hours represents the *minimal* and *maximal* number of hours that participants, i.e., worker, requester, and platform, can spend on crowdsourcing platforms. While legal tools are currently being investigated, e.g., a *Universal Labour Guarantee*[9], there is a stringent need for technical tools allowing official institutions to enforce regulations.

Most current crowdsourcing platforms are independent of each other. However, the emergence of more complex tasks and novel requirements for both workers and requesters, on one hand, and the enforcement of legal regulations, on the other hand, highlights the need for collaboration between crowdsourcing platforms, thus resulting in *multi-platform crowdsourcing systems*. For example, many drivers work for both Uber and Lyft concurrently[10], while requesters may also request multiple drivers from both Uber and Lyft concurrently. The observation holds also for microtask platforms[11], where a common combination among workers is *Amazon Mechanical Turk* and *Prolific*, or for on-demand services[12]. Participants in a crowdsourcing task may also behave maliciously or act as adversaries for their benefits, e.g., violate the privacy of participants or the regulations. Therefore, to check the enforcement of legal regulations in a multi-platform crowdsourcing environment, we need to reconcile *transparency* with *privacy*. Indeed, while enforcing limits on the hours of work over several crowdsourcing platforms requires the *transparent* sharing of information about the crowdsourcing tasks performed by each platform, without any *privacy protection* measures, this may lead to out-of-control disclosures about the participants. Transparent and privacy-preserving collaboration between multiple platforms might also be needed to address *complex cross-platform* tasks. If a requester submits a task with a specified number of requested solutions to multiple platforms, the involved platforms need to collaborate with each other in order to assign workers and provide the specified number of solutions. As a result, a multi-platform system needs to establish consensus between platforms to enable them either to enforce legal regulations or to process

zation, 1919.

9. Future of Work, *op. cit.*

10. For example, `ridesharapps.com` provides tutorials to help drivers manage apps to optimize their earnings `https://rideshareapps.com/drive-for-uber-and-lyft-at-the-same-time/`.

11. Berg, Furrer, et al., *op. cit.*

12. See, e.g., `https://tinyurl.com/nytgigmult`.

cross-platform tasks.

In this chapter, we present *SEPAR*, a technical solution to the problem of imposing global *constraints* on distributed independent entities in the context of multi-platform crowdsourcing systems. The problem is non-trivial because of the complexity of the conjunction of the required properties:

1. **Expressibility**: The constraints need to be expressed in a *simple and non-ambiguous* manner.
2. **Transparent and Privacy-preserving Constraint Enforcement**: Crowdsourcing platforms need to *share information* about the tasks performed without jeopardizing the privacy of participants in order to allow both the enforcement of the global constraints and the collaborative processing of cross-platform tasks.
3. **Distributed Collaboration**: Crowdsourcing platforms are naturally distributed and need to collaborate through distributed consensus algorithms.

SEPAR proposes a privacy-preserving token-based system where global constraints are modeled using *lightweight and privacy-preserving tokens* distributed to workers, platforms, and requesters. Our system formally guarantees that global constraints are satisfied *by construction* and limits the information shared among platforms and participants to the minimum necessary for performing the tasks against *adversarial participants acting as covert adversaries*. We extend our token-based system to allow participants to prove to external entities (e.g., social security agencies) their involvement in crowdsourcing tasks. The resulting proofs are called *certificates*. To provide transparency across multiple platforms, SEPAR proposes a *blockchain-based distributed ledger* shared across platforms. Nonetheless, for the sake of privacy and to improve performance, the blockchain ledger is *not maintained* by any single platform and each platform maintains only a view of the ledger. We then design a suite of distributed consensus algorithms across platforms for coping with the concurrency issues inherent to a multi-platform context and formally prove their correctness. Salient features of SEPAR include the simplicity of its building blocks (e.g., usual asymmetric encryption scheme) and its compatibility with today's platforms (e.g., it does not jeopardize their privacy requirements about requesters and workers for enforcing the regulation).

In a nutshell, the contributions of this chapter are as follows:

1. A privacy model stating formally the privacy requirements of a multi-platform regulated crowdsourcing system based on the well-known simulation paradigm,

2. A simple language for expressing *global constraints*, e.g., limits on the number of work hours, and mapping them to `SQL` constraints to ensure semantic clarity,
3. SEPAR, a *privacy-preserving transparent* multi-platform crowdsourcing system that enforces a given set of constraints. (1) **Privacy** is ensured using *lightweight and privacy-preserving tokens*, while (2) **transparency** is achieved using a *blockchain* shared across platforms,
4. A suite of distributed consensus protocols, and
5. A formal security analysis and thorough experimental evaluation.

This work will be submitted soon to the SIGMOD conference [13].

The chapter is organized as follows. The technical background and related work are discussed in Section 3.2. Section 3.3 defines the problem that SEPAR addresses. The language for expressing constraints is developed in Section 3.4. The token-based system for enforcing constraints and the extended system for certificates are described in Section 3.5. The blockchain ledger and consensus protocols are presented in Section 3.6. Section 3.7 details our thorough experimental evaluation, and finally, Section 3.8 concludes the chapter.

3.2 Related work

In this section, we present relevant related work from crowdsourcing and blockchain literature.

3.2.1 Decentralized systems

In the context of blockchains, Hyperledger Fabric [14] ensures the confidentiality of data using Private Data Collections [15]. Private Data Collections manage confidential data that two or more entities want to keep private from other entities. The underlying idea is that data is sent only to authorized entities, while others only manipulate a hash of data. Quorum [16], as an Ethereum-based [17] blockchain, supports two types of public and private

13. Amiri, Duguépéroux, et al., *op. cit.*
14. Elli Androulaki, Artem Barger, et al., « Hyperledger fabric: a distributed operating system for permissioned blockchains », *in*: *Proc. of EuroSys'18*, 2018, pp. 1–15.
15. *Private Data Collections: A High-Level Overview*, https://www.hyperledger.org/blog/2018/10/23/private-data-collections-a-high-level-overview.
16. JP Morgan Chase, *Quorum white paper*, 2016.
17. *Ethereum blockchain app platform*, https://www.ethereum.org, 2017.

transactions and ensures the confidentiality of private transactions using zero-knowledge proofs (that make it possible to prove statements about data without revealing the data itself).

Providing privacy as well as untraceability in payments has been addressed by ZCash [18]. It also relies on zero-knowledge proofs, which are used to store proofs of ownership of money and to transfer it to someone else, without revealing participants to external parties. Hawk and Raziel [19] [20] manage wider issues, and include general *smart contracts*: these "contracts" allow arbitrary programs to be computed, as opposed to money systems, which only handle a very limited set of operations. The solution proposed relies on a party performing the computation, and providing zero-knowledge proofs of that its execution is correct.

These structures are fully decentralized, which may come as an asset in many contexts, but prevent them from any direct application for crowdsourcing, in which platforms play an important role. Furthermore, these structures are not designed to allow control from an external entity, which may also be an issue for legislators.

Solidus [21] proposes to privately manage a multi-platform banking system, with distinct banks each managing their clients, while allowing cross-platform transactions. In this system, if Alice uses bank A to send money to Bob in bank B, bank A does not learn the identity of Bob (only that Alice transfers money to someone in bank B), nor does bank B learn the identity of Alice (only that someone from bank A sends money to Bob), and any other bank or user cannot learn the identity of Alice, Bob, or the amount of the transaction. However, all can check that the transaction was correct (*e.g.* no-one spent money that they did not own). This mixture of transparency and privacy is achieved by managing, in a public ledger (*e.g.* a blockchain), a complex ORAM-like structure (ORAM [22] being a family of techniques that aim at allowing private download and modification of data on an external server), while letting banks locally manage their users with cleartext values. The architecture of this solution appears to be close to what we attend to provide. However,

18. Daira Hopwood, Sean Bowe, et al., « Zcash protocol specification », *in*: *GitHub: San Francisco, CA, USA* (2016).

19. Ahmed Kosba, Andrew Miller, et al., « Hawk: The blockchain model of cryptography and privacy-preserving smart contracts », *in*: *Proc. of SP'16*, IEEE, 2016, pp. 839–858.

20. David Cerezo Sánchez, « Raziel: Private and verifiable smart contracts on blockchains », *in*: *arXiv preprint arXiv:1807.09484* (2018).

21. Ethan Cecchetti, Fan Zhang, et al., « Solidus: Confidential distributed ledger transactions via PVORM », *in*: *Proc. of SIGSAC CCS'17*, 2017, pp. 701–717.

22. Emil Stefanov, Marten Van Dijk, et al., « Path ORAM: an extremely simple oblivious RAM protocol », *in*: *Proc. of SIGSAC CCS'13*, 2013, pp. 299–310.

allowed operations (spending and receiving money) are far too limited compared to what could be expected in the context of crowdsourcing.

3.2.2 Global crowdsourcing processes

To take into account the overall crowdsourcing process, it is necessary to take into account the reward and its policy. It is clear that all workers want the highest reward, but also that requesters do not want to reward spammers, or tasks that have not been correctly performed. To avoid this issue, it is possible to have the requesters formalize a *reward policy* available to workers, as proposed in Zebralancer [23] and zkCrowd [24]. This policy will define the conditions in which workers can be paid or not, and the amount of the payment. For instance, the reward can depend on some test questions, or only given to answers close to the mean answer. In these systems, the satisfaction of this reward policy is then ensured by using blockchain structures to enable transparency, along with zero-knowledge proofs to maintain privacy, that ensure that the actual payment satisfy the announced policy.

The idea is the following: first, when submitting a task, the requester adds a reward policy, along with the maximum possible reward to the blockchain (this money will not go back to requesters unless they are legitimate to get it back). Contributions of workers to the task are sent in two steps: first, to the requester, and then, encrypted, to the blockchain. To end the process, the requester will analyze the contributions according to her policy, and submit a reward proposition (*e.g.* give \$3 to the first contribution, and \$5 to the second one, etc.), together with a zero knowledge proof that (1) the contributions match the encrypted contributions on the blockchain, and (2) the rewards respect the reward policy.

This proof can then be verified on the blockchain, and payment is made this way, without revealing the identity of workers nor their contributions, and preventing the risk of requesters not rewarding their workers.

It is noticeable that, while the protocol proposed by zkCrowd [25] is more efficient than Zebralancer [26], spammers are avoided in the latter by making use of a *Registration Authority*. This Registration Authority will provide keys to workers, which will be used

23. Lu, Tang, and Wang, *op. cit.*
24. Saide Zhu, Zhipeng Cai, et al., « zkCrowd: a hybrid blockchain-based crowdsourcing platform », *in*: *IEEE Transactions on Industrial Informatics* (2019).
25. *Ibid.*
26. Lu, Tang, and Wang, *op. cit.*

to prevent workers from contributing twice in the same task (without endangering their identity nor enable linkability between their contributions). However, neither of these works provide a way to implement external regulations, or to manage multiple platforms.

3.3 Problem formulation

3.3.1 Motivating example

Multi-platform crowdsourcing systems face two main privacy preserving challenges: enforcing multi-platform regulations and supporting cross-platform tasks. We consider constraining the number of work hours in a ridesharing use-case to illustrate the challenge of enforcing privacy preserving multi-platform regulations. In ridesharing scenarios, a set of workers (i.e., drivers) gives rides to a set of requesters (i.e., travelers) through a set of platforms, e.g., Uber, Lyft, Curb, and Juno, where each driver (resp. traveler) registers to one or more platforms. Regulations on the hours of work often specify minimal and maximal number of work hours that can be performed by the participants. For instance, (1) the total work hours of a driver per week may not exceed 40 hours to follow the *Fair Labor Standards Act*[27] (FLSA), (2) a driver has to work at least 5 hours per week to be eligible for insurance coverage, and (3) the total work hours of all drivers on a platform should be at least 1000 hours per week to enable the platform to fill for a tax refund. A multi-platform crowdsourcing system needs to express and enforce such regulations while preserving the privacy of participants. Indeed, the system needs to (1) provide a technical tool to enable official institutions expressing the regulations, (2) support transparent sharing of information about the crowdsourcing tasks performed by each platform to enable them checking the enforcement of regulations, and (3) preserve the privacy of participants.

Supporting complex cross-platform tasks that may need multiple contributions from possibly different platforms raises the second set of challenges. For instance, a requester who has registered with *Amazon Mechanical Turk*, *Appen* and other microtask platforms might need hundreds or thousands of contributions at the same time. The requester would like to accept these contributions from workers regardless of the platforms the microtasks are performed on. Since workers from different platforms might want to perform these contributions, the system needs to establish consensus among the various microtask platforms to assign workers and provide the specified number of solutions without revealing

27. `https://www.dol.gov/agencies/whd/flsa`

any private information about the workers to competing platforms.

3.3.2 Crowdsourcing environment

Participants

Today's realistic *crowdsourcing environments* consist of a set of workers $\mathcal{W}$ interacting with a set of requesters $\mathcal{R}$ through a set of competing platforms $\mathcal{P}$. We call *participants* the workers, platforms, and requesters of a crowdsourcing environment. Each worker $w \in \mathcal{W}$ (1) registers to one or more platforms $\mathcal{P}_w \subset \mathcal{P}$ according to her preferences and, through the latter, (2) accesses the set of tasks available on $\mathcal{P}_w$, (3) submits each *contribution* to the platform $p \in \mathcal{P}_w$ she elects, and (4) obtains a *reward* for her work. On the other side, each requester $r \in \mathcal{R}$ similarly (1) registers to one or more platforms $\mathcal{P}_r \subset \mathcal{P}$, (2) issues a *submission* which contains her tasks $\mathcal{T}_r$ to one or more platforms $p \in \mathcal{P}_r$, (3) receives the contributions of each worker w registered to $\mathcal{P}_r \cap \mathcal{P}_w$ having elected a task $t \in \mathcal{T}_r$, and (4) launches the distribution of rewards. Platforms are thus in charge of facilitating the intermediation between workers and requesters. A *crowdsourcing process* π connects three parties – a worker w, a platform p, and a requester r – with each other and aims to solve a task $t \in \mathcal{T}_r$ through p, and consists in the steps (2) to (4) above. Figure 3.1 shows a crowdsourcing infrastructure with four platforms, four workers and four requesters.

In this work, we do not focus on the description of tasks and contributions and consequently model both as arbitrary bitstrings $\{0,1\}^*$ and make no assumption on the distribution of rewards to workers [28].

Finally, workers, requesters, and platforms are all equipped with the cryptographic material required by SEPAR: a pair of usual public/private asymmetric keys (e.g., RSA) and a pair of public/private asymmetric *group* keys [29] where the union of all workers forms a group (in the sense of group signatures), similar to the union of all requesters, and to the union of all platforms. Participants acquire them when joining SEPAR (see Section 3.5 for more information).

28. A task embeds all the information necessary to be performed by a worker (*e.g.,* the precise description of the work that must be performed, a *reward policy* for distributing the reward among contributors, the expected number of contributions).

29. Jan Camenisch and Jens Groth, « Group signatures: Better efficiency and new theoretical aspects », *in*: *Proc. of SCN'04*, Springer, 2004, pp. 120–133.

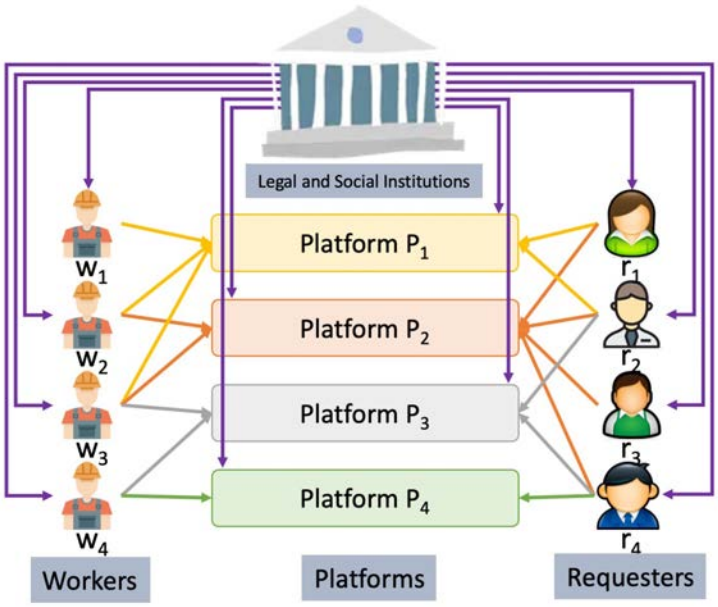


Figure 3.1 – A crowdsourcing infrastructure

Interactions with institutions

Crowdsourcing environments do not exist in a vacuum but rather are integrated within society as a whole. The participants need in particular to interact with legal institutions (in order to enforce the local labor laws) and with social institutions (in order to enable local social rights). We capture these interactions through less-than *constraints* and greater-than *certificates*.

Constraints. A set of constraints embodies the labor policy that applies to a given crowdsourcing environment. Essentially, a constraint expresses a limit on the *actions* that can be performed by the participants of the crowdsourcing environment, e.g., the total working hours of a worker per week must not exceed 40 hours across all platforms. Constraints must be expressed in an intuitive language that is both expressive enough to adapt to a variety of real-world policies and at the same time restrictive enough to guarantee the efficiency of their enforcement.

Certificates. A certificate is a piece of information that participants can provide to third parties to prove that they took part in a given crowdsourcing process. Contrary to constraints, they are not *a priori* specification: they are made available during the process to the participants involved and can be provided by participants to other parties on demand after the process. Certificates are well suited to real-world situations such as

conforming to legal obligations, suing other parties in court in case of abuse, or legitimizing applications to grants or tax refund depending on local legislation, e.g., a driver has to work at least 5 hours per week to be eligible for insurance coverage or the total work hours of all drivers on a platform should be at least 1000 hours per week to enable the platform to fill for a tax refund.

Distribution of Platforms

We do not make any assumptions on the inner working of platforms, especially on their inner implementation of crowdsourcing processes (e.g., task assignment algorithm, workers contributions delivery). However, we stress that our approach is compatible with distributed infrastructures, supported by one or more data centers, following today's fault tolerance and performance standards. In particular, we assume each platform consists of a set of nodes in an asynchronous distributed network where transactions are replicated on the distributed persistent transparent datastore of each node. In this chapter and due to the unique features of blockchains such as transparency, provenance, and fault tolerance, the datastore is implemented using a *blockchain*.

A crowdsourcing environment processes *internal*, i.e., submitted to a single platform, and *cross-platform*, i.e., submitted to more than one platform, tasks. Processing a task (either internal or cross-platform) requires agreement from the nodes of the involved platforms. To establish agreement among the nodes, we introduce *local* and *cross-platform* consensus protocols. In addition, we enable all platforms checking the satisfaction of constraints by establishing consensus among every node of all platforms. To do so, a *global* consensus protocol is introduced.

3.3.3 Security model

We consider that any participant in a crowdsourcing environment may act as a *covert adversary*[30] that aims at inferring anything that can be inferred from the execution sequence and that is able to deviate from the protocol if no other participant detects it. Adversarial participants may additionally collude.

The privacy definition that we adopt requires that no participant obtains or infers any information about a crowdsourcing process beyond what is strictly needed for accomplishing

30. Yonatan Aumann and Yehuda Lindell, « Security against covert adversaries: Efficient protocols for realistic adversaries », *in*: *Proc. of TCC'07*, Springer, 2007, pp. 137–156.

its local crowdsourcing processes and for the distributed enforcement of constraints. We formalize below this requirement by defining the set of *secrets* and by using the well-known simulatability model often used by secure multi-party computation algorithms.

Consider a crowdsourcing process π between worker w, platform p, and requester r for solving task t. The information generated by the execution of π consists at least of a starting event `BEGIN`, an ending event `END`, and the relationship between the three participants (w, p, and r) with the task t. We denote it by the 6-tuple $(\texttt{BEGIN}, \texttt{END}, w, p, r, t)$. The {`BEGIN`, `END`} events are abstract representations of the information that π is starting or ending. They may be given, e.g., by the exchange of messages between the participants to π, and may come with additional concrete information (e.g., timestamps, IP address). Our privacy definition focuses on *workers, requesters, and tasks* and requires the secrecy of the corresponding parts of this tuple from the participants that are not involved in π. However, since a given task t may be submitted to several platforms and then be accessed by several workers, the platforms and workers not involved in π but that receive t still need to learn that t has been completed (e.g., to manage their local copy of the task). We capture this subtlety through a *varying set of disclosures*, denoted δ^π, plugged into a *unified simulation-based privacy definition*. Our definition does not leak any information about the worker and the requester involved in π when it is not needed by π. It tolerates the disclosure of the {`BEGIN`, `END`} events and of the platform p to all participants, whatever their involvement in π. This allows platforms to share information for enforcing global regulations (e.g., check that all participants satisfy the related constraints before executing π), and to collaborate for managing correctly the cross-platforms tasks. Note that for simplicity we will use the same notation δ for disclosures concerning *sets* of crowdsourcing processes as well.

We specify the set of disclosures as follows:

— Secrecy against the participants that are **not involved** in π and that have **not received** task t from requester r: they must not learn anything about the worker, the task, and the requester involved in π:

$$\delta^\pi_{\neg R \neg I} = (\texttt{BEGIN}, \texttt{END}, p)$$

— Secrecy against the platforms and workers that have **received** the task t from r but that are **not involved** in π: they must be aware that t has been performed (e.g., for not contributing to t) but must not know that it has been performed by

worker w:

$$\delta^{\pi}_{R\neg I} = (\texttt{BEGIN}, \texttt{END}, p, r, t)$$

— Secrecy against participants that are directly **involved** in π (and have thus **received** task t): the information about the execution of π disclosed to w, p, and r is complete simply because they run π [31]:

$$\delta^{\pi}_{RI} = (\texttt{BEGIN}, \texttt{END}, w, p, r, t)$$

We are now ready to state the privacy definition.

Definition 12. Let Π be a set of crowdsourcing processes executed by σ an instance of SEPAR over a set of participants. We say that σ is δ^{Π}-*private* if, for all $\pi \in \Pi$, for all computationally-bounded adversaries $\texttt{A}$, the sets of disclosures $(\delta^{\pi}_{\neg R\neg I}, \delta^{\pi}_{R\neg I}, \delta^{\pi}_{RI})$, arbitrary background knowledge $\chi \in \{0,1\}^*$, the distribution representing the adversarial knowledge over the input dataset in the real setting is *computationally indistinguishable* from the distribution representing the adversarial knowledge in an ideal setting in which a trusted third party $\texttt{cp}$ executes the crowdsourcing process Π of σ:

$$\texttt{REAL}_{\sigma,\texttt{A}(\chi,\delta^{\pi}_i)}(\mathcal{W},\mathcal{P},\mathcal{R},\mathcal{T}) \stackrel{c}{\equiv} \texttt{IDEAL}_{\texttt{cp},\texttt{A}(\chi,\delta^{\pi}_i)}(\mathcal{W},\mathcal{P},\mathcal{R},\mathcal{T})$$

where $i \in \{\neg R\neg I, R\neg I, RI\}$, and $\texttt{REAL}$ denotes the adversarial knowledge in the real setting and $\texttt{IDEAL}$ its counterpart in the ideal setting.

3.3.4 Problem

We address the problem of designing the SEPAR system in charge of allowing the execution of crowdsourcing processes (1) while guaranteeing together the correctness of the constraints defined by external institutions and the privacy of participants against covert adversaries, (2) over distributed crowdsourcing platforms that communicate through the `local`, `cross-platforms`, and `global` consensuses.

31. There exist cases where the platform acts as a proxy between a worker and a requester such that one side has no information on the other side. Our privacy definition does not require that, but is compatible with more stringent secrecy.

3.4 Expressing global regulations

3.4.1 Expressing constraints

Syntax. We define a constraint c as being essentially (1) a triple (w, p, r) that associates a worker w, a platform p, and a requester r, and (2) a threshold θ (an integer) that defines the upper bound of c [32]. Intuitively, a constraint $((w, p, r), \theta)$ states that there must not be more than θ actions between the worker w, the platform p, and the requester r (see below for the detailed semantics). We also allow two wildcards to be written in any position of a triple: $*$ and $\forall$. First, the $*$ wildcard allows to ignore one or more elements of a triple [33]. For example $(*, p, r)$ means that the constraint applies to the couple (p, r). A triple may contain up to three $*$ wildcards. An element of a triple that is not a $*$ wildcard is called a *specified participant* of the constraint. Second, the $\forall$ wildcard factorizes the writing of triples because it allows to express a constraint that must hold for all participants in the same group of participants [34]. For example, $(\forall, p, r)$ represents the following set of triples: $\{(w, p, r)\}, \forall w \in \mathcal{W}$. We denote $\mathcal{C}$ the complete set of constraints.

Semantics. We give now a precise definition of the semantics of our constraints by illustrating how they translate to `SQL` constraints. Let assume that there exists a table of actions `A` that records all actions performed between any triple of worker, platform, requester. The attributes of `A` are `WORKER`, `PLATFORM`, `REQUESTER`. For simplicity, we consider a constraint c without any wildcard, i.e., $c \leftarrow ((w, p, r), \theta)$. The semantics of c is the same as the following `SQL` query :

```
ALTER TABLE A ADD CONSTRAINT c CHECK (
  NOT EXISTS (
    SELECT * FROM A
    WHERE WORKER=w AND PLATFORM=p AND REQUESTER=r
    GROUP BY WORKER, PLATFORM, REQUESTER
    HAVING COUNT(*) ≥ θ
  ) );
```

32. Extending constraints with, e.g., labels for defining categories of actions (e.g., working time) or validity periods (e.g., "one week", "one month"), is straightforward.
33. Intuitively, the $*$ wildcard means "whatever".
34. Intuitively, the $\forall$ wildcard means "for each".

The presence of a $*$ wildcard in the triple simply leads to removing the corresponding attributes in the `WHERE` and `GROUP BY` clauses. The presence of a $\forall$ wildcard leads to expanding it to the set containing all the elements that it represents (e.g., all workers if the $\forall$ wildcard is at the first position in the triple) and to generate the cartesian product between the resulting set and the elements at the two other positions of the triple (that may be $\forall$ wildcards as well). Finally, the semantics of a set of constraints is the conjunction of the constraints contained in the set.

Example. the weekly FLSA limit on the total work hours per worker can easily be expressed as $c_{FLSA} \leftarrow ((\forall, *, *), 40)$.

3.4.2 Expressing requests for certificates

Syntax and Semantics. Certificates allow a participant called *prover* (e.g., worker) to prove to an external entity called *verifier* (e.g., social security agency) that a minimal number of hours have been spent on crowdsourcing platforms (e.g., for applying to insurance coverage). *Requests for certificates* (e.g., from social security agencies) are expressed using the same syntax as the constraints with the following two differences. First, the θ threshold does not represent an upper bound on actions that cannot be exceeded, but a lower bound on actions that have to be proved. And second, there must always be at least one specified participant in a request for certificates, i.e., typically the prover. This syntax allows verifiers to follow *minimal disclosure principles* by requesting from the prover exactly the information needed about the crowdsourcing processes performed. There is no need to request the identities of the participants with whom the prover collaborated. Additionally, it is trivial to connect multiple requests for certificates through conjunctions and disjunctions if needed.

Examples. A social security institution can request each worker w applying for insurance coverage to prove that she worked in total more than 5 hours: $r_1 \leftarrow ((w, *, *), 5)$ is both necessary and sufficient. Similarly, the request $r_2 \leftarrow ((*, p, *), 1000)$ allows a tax institution to ask for each platform p applying for a tax refund to prove that the total work hours of all its workers is at least 1000 hours.

3.5 Enforcing global regulations

In this section, we develop our conception of constraints and certificates, how they are built and how to use them, and we prove the correctness and the privacy guarantees of our construction.

3.5.1 Implementing a token-based system

In this section we show how constraints and certificates, which are expressed in Section 3.4, can be enforced and produced respectively. Inspired by e-cash systems, we enforce constraints and produce certificates by managing two *budgets* per participant while preserving both the privacy of participants and the correctness of budgets. Our proposal makes use of a centralized authority, called the *registration authority (RA for short)*. RA registers the participants to the crowdsourcing environment, sends them the required cryptographic material, receives the set of constraints, and manages the budgets. The required cryptographic material includes a pair of public/private asymmetric keys (e.g., RSA) and a pair of public/private asymmetric *group* keys [35] for which the registration authority is the group manager, while the set of constraints may be expressed by the regulators through a dedicated interface. We instantiate the budgets based on labeled, single-use, privacy-preserving *tokens* and use a persistent transparent datastore to guarantee their correct and validated spending by participants. The persistent datastore is implemented using a *blockchains*. To process crowdsourcing tasks, our token-based system is defined by five functions: `GENERATE` for initializing the budgets and refilling them, `SPEND` for spending portions of the budgets, `PROVE` for providing certificates to a third party, `CHECK` for checking whether a given spending is allowed or not, and `ALERT` for reporting dubious spending.

The GENERATE function

The registration authority uses the `GENERATE` function to initialize the budgets, i.e., constraint and certificate tokens of all participants (i.e., workers, platforms, requesters) and refill them periodically [36] according to the set of constraints $\mathcal{C}$ to enforce.

35. Camenisch and Groth, *op. cit.*

36. The refreshment rates of budgets is easily computed from the validity periods of constraints (see Section 3.4).

Constraint tokens. For each constraint $c \leftarrow ((w, p, r), \theta)$, the registration authority generates θ tokens and sends a copy of each token to each specified participant of c. A token consists of a public and a private component. The *public component* is a pair made of a *number used only once* (referred to as a *nonce* below) generated by the registration authority and a signature of the nonce produced by the registration authority [37]. The public component will be used later (upon completion of the corresponding task) by other platforms to check the validity of tokens. The *private component* is an index allowing the participants involved in a crowdsourcing process to select the correct set of tokens given the other specified participants involved in the process. We implement this private component as a list containing the public keys of the specified participants (in the corresponding constraint) [38]. Let $tk^{\dagger}$ be a constraint token, $tk^{\dagger}_{pub}$ be its public component, $tk^{\dagger}_{priv}$ be its private component, N be a nonce and *pubs* the list of public keys. The constraint token is thus the couple $(tk^{\dagger}_{pub}, tk^{\dagger}_{priv})$ where $tk^{\dagger}_{pub} = (N, Sign(key_{priv,RA}, (N)))$ and $tk^{\dagger}_{priv} = pubs$.

Certificate tokens. Certificate tokens are generated initially by the registration authority for all participants. For each crowdsourcing process, a single certificate token is linked to a fully specified triplet of participants (w, p, r). The number of certificate tokens produced is decided initially, but their quantity is not as easy to decide as for constraint tokens since it is not capped by a θ threshold. For simplicity, we assume that there is at least one constraint in the system, and the smallest threshold for all constraints is θ_{min}. Then, θ_{min} is a sufficient upper bound of the number of crowdsourcing processes in which any given triplet of participants is involved. It is therefore enough to produce $\theta_{min} \times |\mathcal{W}| \times |\mathcal{P}| \times |\mathcal{R}|$ certificate tokens. In practice, the number of tokens produced can be drastically reduced in a straightforward manner by letting participants declare to RA the subset of participants they may work with (*e.g.,* selecting a subset of platforms, or domains of interest).

As stated above, a certificate token always relates to a fully specified triplet (w, p, r) and to its owner o (i.e., one of the participants in the triplet). Similar to a constraint token, it consists of a public and a private component. The public component consists of a nonce as well as the signature of the nonce produced by the registration authority. The private component, on the other hand, is a triplet in which each element certifies (i.e., signs) the

37. Extending tokens with labels and/or timestamps for supporting the validity periods of constraints is straightforward.

38. The use of a public key generated by the registration authority is important here because (1) it can be shared among participants without disclosing their identities, i.e., it is a pseudonym, (2) the corresponding private key can be used by participants for mutual authentication in order to guarantee the correctness of the index and consequently of the choice of tokens.

association between the owner o and another specified participant. More formally, let tk^* be a certificate token, tk^*_{pub} be its public part, tk^*_{priv} be its private part, N be a nonce, o be the identity of the participant owner of the token, and (w, p, r) be the related triplet. The certificate token is thus the pair $(tk^*_{pub}, tk^*_{priv})$ where $tk^*_{pub} = (N, Sign(key_{priv,RA}, (N)))$ and $tk^*_{priv} = (Sign(key_{priv,RA}, (N, o, w)), Sign(key_{priv,RA}, (N, o, p)), Sign(key_{priv,RA}, (N, o, r)))$.

The SPEND function

Requesters create and send their tasks to a platform and the platform submits the tasks in either its own datastore (for local tasks) or all involved platforms (for cross-platform tasks). Once the task is published, the workers can indicate their intent to perform the task by sending a *contribution intent* to their platforms. If a contribution is still needed for the task, the `SPEND` function is performed as follow. First, for a given constraint $c \in \mathcal{C}$, the platform requests the public component of a constraint token corresponding to c from the *initiator* (one of the specified participants in constraint c). For certificates, the platform is the initiator. The platform includes the task, an identifier for the contribution (e.g., a nonce generated by the platform), and a signature of the identifier concatenated to the task in its request message. Therefore, workers and requesters will be able to prove that they were asked for tokens, even if the platform fails. The initiator then chooses a token to spend and sends it to the platform. Once the platform receives the required token, it sends the public component of the constraint token to all the specified participants. For certificates, the platform sends the public part of the certificate tokens to all participants of the process. The platform also requires them to send back two signatures: (1) the group signature of the token (which will be later verified by all platforms, together with the token, when it is shared with all platforms), and (2) the group signature of the pair consisting of a token and a task. Note that the second signature, while not revealing the task by itself, can be used by participants to verify that tokens are used on the task they are intended to be used on. Again, this demand is associated with the task and the identifier of the contribution, and is signed by the platform.

Finally, for each task, a transaction consisting of all spent constraint tokens from each specified participant (all spent certificate tokens from every participant in the case of certificates) is committed to the datastore of all platforms. For each token, the transaction includes first, the public component of each token, second, the group signature of the public component of each token (i.e., for a constraint token $tk^\dagger$: $GroupSign(key_{priv,part}, Group, tk^\dagger_{pub})$),

and third, the group signature of the public part of each token together with the associated task t (i.e., for a constraint token $tk^{\dagger}$: $GroupSign(key_{priv,part}, Group, (t, tk^{\dagger}_{pub}))$).

The PROVE function

The `PROVE` function is used by participants to provide certificates to a third party. The use of certificate tokens is relatively straightforward. During the crowdsourcing processes, participants store the private components of certificate tokens which will be used later to deliver certificates on demand. A participant indeed initiates the `PROVE` function by sending the related subpart(s) of the private component of the corresponding tokens to the verifier. As an example, for a $((w, *, *), 5)$ request for certificates, the worker w sends the subparts containing w from the private parts of all 5 certificate tokens. The verifier, first, checks the signature of the registration authority to verify that the participant was involved in the task, and then, checks the nonce stored in the datastore to ensure that the token has been shared and validated by all platforms.

The CHECK and ALERT functions

The `CHECK` and `ALERT` functions are used to detect and report either the malicious behavior of participants resulting in an invalid consumption of tokens or the failure of a platform. The complete set of verifications protects against (1) the forgery of tokens (verification of the signatures), (2) the replay of tokens (verification of the absence of double-spending), (3) the relay of tokens (verification of the absence of usurpation), and (4) the illegitimate invalidation of tokens (timeout against malicious platform failures). The first two verifications are straightforward and performed during the global consensus. We explain the last two verifications.

Usurpation. When a token is appended to the datastore of all platforms, anyone (whether involved in the corresponding crowdsourcing process or not) can `CHECK` its nonce. If a participant detects a nonce that was received from the registration authority but not spent [39], she `ALERT`s the registration authority. The registration authority will reveal the corresponding participant in the group signature (e.g., the worker's group signature if the alert comes from a worker), and checks whether it has been signed by the same participant that sent the token. Similarly, if a participant detects that a token has not been spent

39. For example, a platform p can collude with a worker w_1 to spend a token dedicated to a $(w_2, p, *)$ constraints.

on the right task, she ALERTs the registration authority. After an alert, the registration authority has to act either against the target participant of the alert (true positive) or against the participant originating the alert (false positive). The possible actions (e.g., ban the participant) depend on the context.

Platform failure. If a platform fails after it requested tokens or signatures and does not recover (e.g., tokens are not appended to the datastore), the tokens revealed to the platform are lost: they cannot be used in any other crowdsourcing process because they are not as protective as intended (i.e., the platform knows the association between them and the corresponding participants), and they are not spent either. In that case, workers or requesters send an ALERT to the registration authority including (1) the identifier of the platform, (2) the identifier of the task, and (3) all the requests received from the platform. The registration authority then checks whether the number of requests sent by the platform for the given task matches the corresponding number of messages committed in the datastore. If there are more requests, the registration authority sets a timeout (e.g., to let unfinished transactions end or the platform recover from a failure). When the timeout is over, the registration authority can act against the platform.

3.5.2 Task processing sequence

In summary, five main phases exist during the processing of a crowdsourcing task: (1) initialization, (2) publication, (3) assertion, (4) verification, and (5) execution.

Initialization. The registration authority provides all parties with their keys and tokens.

Publication. Requesters create and send their tasks to platforms. If a requester wants to publish its task on more that one platform (i.e., a cross-platform task), the involved platforms collaborate with each other to create a common instance of the task. The involved platforms then publish the tasks on their datastores through submission transactions and inform their workers in their preferred manner for accessing tasks.

Assertion. After a worker has retrieved a task, the worker sends a *contribution intent* message to the platform without revealing the actual contribution. The platform then updates the number of required contributions for the task and publishes the contribution intent in its datastore through a claim transaction. For cross-platform tasks, the platform informs other involved platforms about the received contribution intent, so that all involved platforms agree with the number (and order) of the received contribution intents (i.e., claim transactions). If the desired number of contribution for the task has been achieved,

the process is aborted. Note that while the requester does not choose the workers, it is possible to enforce a selection with *a priori* criteria, passed through the platform. Another straight-forward enhancement would be to add a communication step, by forwarding the contribution intent, together with the worker's identity to the requester, and letting her approve it or not. This communication, however, requires the disclosure of the worker's identity to requesters even before a contribution is accepted.

Verification. Once the contribution intent has been accepted by the platform(s), the platform asks the corresponding requester and worker to send the required tokens and signatures, through the `SPEND` function, developed in Section 3.5.1. Upon receiving all tokens and signatures, the platform shares them with all platforms and the tokens and their signatures are published to the datastores through verification transactions. From this point, anyone can check the validity of requirements with the `CHECK` function (and `ALERT` if required), as developed in Section 3.5.1.

Execution. Once all parties have checked the validity of the task, its tokens and group signatures, the actual contribution can be given to the requester and reward to the worker through the platform.

A sequence chart of this protocol is provided in Figure 3.2.

3.5.3 Privacy analysis

We show below in the suite of Theorem 9, Lemma 1, Lemma 2, and Theorem 10 that the global execution of SEPAR satisfies the δ^{Π}-*private* model against covert adversaries. First, Theorem 9 restricts the adversarial behavior to inferences (i.e., similar to a *honest-but-curious* adversary) and shows that the execution of SEPAR satisfies δ^{Π}-privacy. Second, we extend the possible behaviors to malicious behaviors aiming at jeopardizing the enforcement of constraints and of requests for certificates, and show that they are systematically detected by SEPAR (Lemma 1 focuses on constraints and Lemma 2 on certificates). This prevents covert adversaries to perform malicious actions, limiting them to inferences. Since Theorem 9 shows that SEPAR is δ^{Π}-private against adversaries restricted to inferences, and Lemma 1 and Lemma 2 show that malicious behaviors are prevented, it follows that SEPAR is δ^{Π}-private against covert adversaries (Theorem 10).

Theorem 9. *(Privacy (inferences)) For all sets of crowdsourcing processes* Π *executed over participants* $\mathcal{W}$*,* $\mathcal{P}$*, and* $\mathcal{R}$ *by an instance of SEPAR* σ*, then it holds that* σ *is* δ^{Π}*-private against covert adversaries restricted to inferences.*

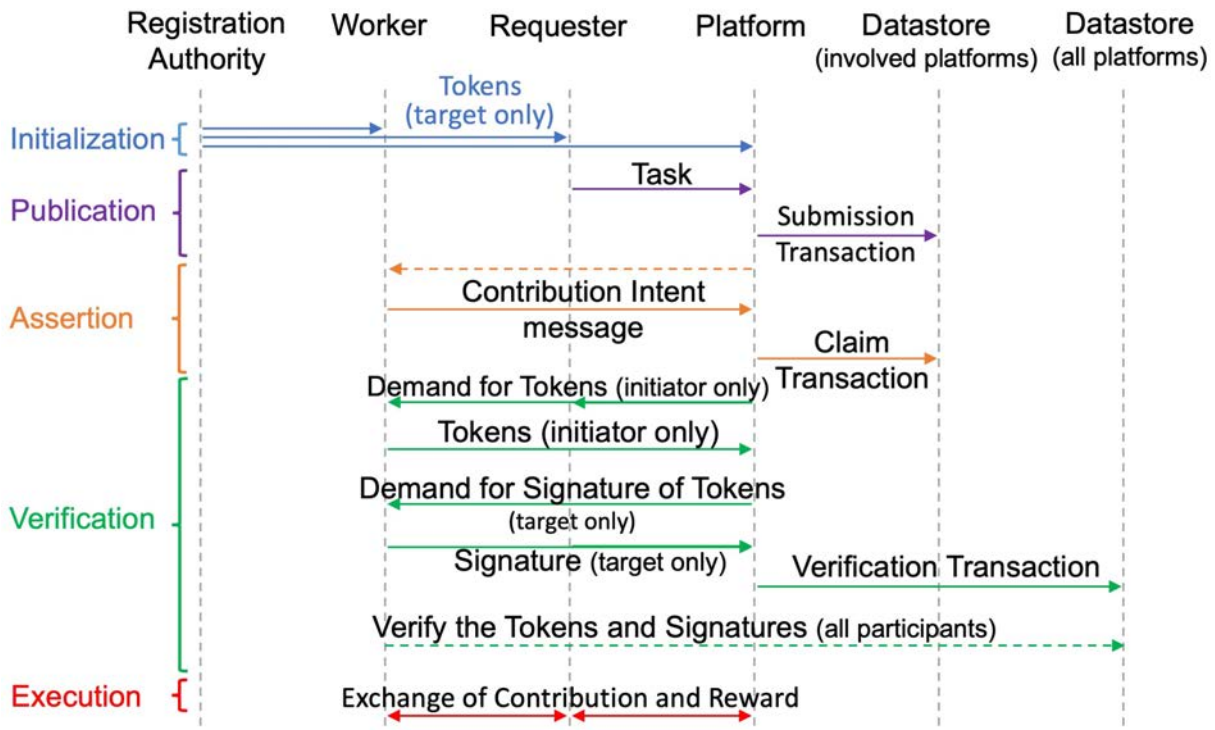


Figure 3.2 – Sequence chart (references to specified participants include all participants for certificate tokens)

(sketch). First, we focus on the content of tokens and show that it is harmless. For each crowdsourcing process π, the information contained within the tokens exchanged and stored in the public datastore is made of (1) the public parts tk_{pub} of the tokens involved (*i.e.,* a nonce and a signature) and (2) of the group signatures of the participants to π. The nonce is generated by the registration authority independently from π, thus do not leak any information about π [40]. Since the group signatures are generated by a semantically-secure group signature scheme, they do not leak anything to the real adversary (computationally bounded) beyond the groups of the signers, which is also available to the ideal adversary.

Second, we concentrate on the information disclosed along the execution sequence of crowdsourcing processes. We consider below each disclosure set δ_i^π in turn and show that the computational indistinguishability requirement between the ideal setting and the real setting (where the instance of SEPAR σ executes Π) is satisfied in all cases.

40. Including a generation timestamp into the public part of tokens, for supporting the validity periods of constraints, would not leak information about π beyond its probable execution timeframe, which is already captured by the $BEGIN$ and END events allowed to leak in the δ^Π-confidentiality model. Indeed, the timestamps would be generated by the registration authority independently from π.

Disclosure set $\delta^{\pi}_{\neg R \neg I}$. The $\delta^{\pi}_{\neg R \neg I}$ disclosure set contains the information allowed to be disclosed to the participants that are not involved in a crowdsourcing process $\pi \in \Pi$ and that have not received the related task (*i.e.,* $(\texttt{BEGIN}, \texttt{END}, p)$). First, we focus on the subset of such participants that are requesters or workers. In the ideal settings, these participants learn nothing beyond the information contained in $\delta^{\pi}_{\neg R \neg I}$. In the real setting, when π is executed by σ, these participants are not involved in any consensus. They are only able to observe the state of the datastore. The latter is updated exactly once for π, when π ends, for storing the tokens spent: only the ending event (`END`) is disclosed, which is already contained within $\delta^{\pi}_{\neg R \neg I}$. Second, we focus on the platforms. In the ideal setting, they are given $\delta^{\pi}_{\neg R \neg I}$. In the real setting, they participate to the global consensus and are able to observe the state of the datastore. Consequently, they learn p from the global consensus (i.e., the platform that initiates the global consensus) and the ending event `END` [41]. Both are already contained within $\delta^{\pi}_{\neg R \neg I}$.

Disclosure set $\delta^{\pi}_{R \neg I}$. The $\delta^{\pi}_{R \neg I}$ disclosure set contains the information allowed to be disclosed to the workers and platforms that have received the task t from r but that are not involved in the crowdsourcing process $\pi \in \Pi$ (*i.e.,* $(\texttt{BEGIN}, \texttt{END}, p, r, t)$). First, we focus on the subset of participants that are workers. In the ideal settings, they learn nothing beyond the information contained in $\delta^{\pi}_{R \neg I}$. In the real setting, we see from the global execution sequence of SEPAR that (1) they are not involved in any consensus, (2) but are able to observe the state of the datastore, (3) have received the task t, and (4) may receive an abort from their platform if they contribute to t while t has already been solved. From (2) and (4), they are able to learn the ending events of crowdsourcing processes, from (3) they learn t, and from (1) they do not learn any other information. As a result, they learn $(\texttt{END}, t)$, which is contained in $\delta^{\pi}_{R \neg I}$. Second, we focus on the subset of participants that are platforms. In the ideal settings, they learn nothing beyond the information contained in $\delta^{\pi}_{R \neg I}$. In the real setting, (1) they receive the tasks from the requesters, (2) they participate to the cross-platform consensuses and to the global consensuses in addition to (3) observing the same information as the workers. From (1) they learn t and r for each process $\pi \in \Pi$. From (2), they learn p, `BEGIN`, and `END` because: the cross-platform consensus discloses the initiating platform and the starting event, and the global consensus discloses the initiating platform together with the ending event. From (3), they learn the ending event

41. The verifications performed by the worker and the requester involved in π are performed on the instance of the datastore stored on p. Indeed, the block resulting from the global consensus contains all the necessary information both for performing the verification and for checking that it results from the global consensus.

and the task. As a result, such platforms learn about all $\pi \in \Pi$ the following information $(\texttt{BEGIN}, \texttt{END}, r, p, t)$, which is exactly $\delta^{\pi}_{R\neg I}$.

Disclosure set δ^{π}_{RI}. The computational indistinguishability requirement between the real setting and the ideal setting is trivially satisfied for the δ^{π}_{RI} set of disclosures because δ^{π}_{RI} contains all the information about the execution of the crowdsourcing process π so σ does not (and cannot) disclose more information. □

Lemma 1. *(Detection of malicious behaviors (constraints)) A crowdsourcing process π executed over participants $\mathcal{W}$, $\mathcal{P}$, and $\mathcal{R}$ by an instance of SEPAR σ, completes successfully without rising a legitimate alert if and only if it does not jeopardize any constraint.*

(sketch). We have to show first that the constraint tokens allocated to participants can be spent, and second that participants cannot spend more.

Participants can spend their tokens. In order to prevent participants from spending their constraint tokens, an attacker has three main possibilities. First, she can try to acquire tokens belonging to another participant and to spend them (i.e., relay attack). However, this rises an `ALERT` with certainty. Indeed, if a token is spent by an illegitimate participant, it is stored on the datastore and is thus accessible to the legitimate participant who is able to detect it through the nonce and to rise an `ALERT` to the registration authority (including the group signatures stored along the token). Second, the attacker (platform only) could try to misuse tokens by spending them in a way that was not intended by the legitimate owner (i.e., relay attack). However, this would be detected by participants as well because the signature of the task would not be valid. Third, the attacker (platform only) may abort the process after having received tokens but before performing the global consensus (i.e., illegitimate invalidation). However, after a timeout, the other involved participants simply send an `ALERT` to the registration authority and prove that their tokens were requested by the platform (signatures of the requests for tokens and signatures), and therefore that the platform behaves illegitimately.

Participants cannot spend more. First, an attacker may produce additional tokens (i.e., forge attack). However, the public parts of tokens must contain valid signatures produced by the registration authority. Second, an attacker may try to spend a token more than once (i.e., replay attack). However, the nonce of a token that must be spent must not already be in the datastore. Finally, an attacker may simply omit sending any token. However, the public parts of tokens are required for the successful completion of the global consensus. □

Lemma 2. *(Detection of malicious behaviors (certificates)) A participant can produce a certificate about a crowdsourcing process π executed over participants $\mathcal{W}$, $\mathcal{P}$, and $\mathcal{R}$ by an instance of SEPAR σ, if and only if (1) she was involved in π and (2) π completes successfully.*

(sketch). First, certificates about a crowdsourcing process π that completed successfully can always be produced by the participants involved in π. Indeed, the certificate tokens are produced by the registration authority and sent to all participants (i.e., the number of certificate tokens is correct), and only the successful crowdsourcing processes store the certificate tokens in the datastore. Second, participants cannot spend a certificate token more than once because the nonce of a token that must be spent must not already be in the datastore. Third, participants cannot produce any certificate token by themselves because their public parts must contain valid signatures produced by the registration authority. □

Theorem 10. *(Privacy (inferences and malicious behaviors)) For all sets of crowdsourcing processes Π executed over participants $\mathcal{W}$, $\mathcal{P}$, and $\mathcal{R}$ by an instance of SEPAR σ, then it holds that σ is δ^{Π}-private against covert adversaries.*

Proof. Theorem 9 shows that the execution of SEPAR satisfies δ^{Π}-privacy against covert adversaries restricted to inferences, while Lemma 1 and Lemma 2 shows that malicious behaviors aiming at jeopardizing the guarantees token-based system are detected and consequently prevented within SEPAR. As a result it follows directly that the execution of SEPAR satisfies δ^{Π}-privacy against covert adversaries. □

3.6 Coping with distribution

SEPAR is a multi-platform crowdsourcing system where multiple globally distributed platforms collaborate with each other to process crowdsourcing tasks. To realize such distributed collaborations and due to the unique features of permissioned blockchains such as transparency and provenance, which are needed by crowdsourcing applications, SEPAR is deployed on a permissioned blockchain to implement the persistent datastore. In this section, we first present the distributed blockchain ledger of SEPAR and then, show how SEPAR establishes consensus on the order of transactions within and across different platforms.

3.6.1 Blockchain ledger

In a blockchain, transactions are recorded in an append-only data structure, called *Blockchain ledger*. The blockchain ledger in SEPAR includes all submission, claim, and verification transactions of all internal as well as cross-platform tasks. To ensure data consistency, an ordering among transactions in which a platform is involved is needed. The total order of transactions in the blockchain ledger is captured by *chaining* the transactions (blocks) together, i.e. each transaction block includes a sequence number or the cryptographic hash of the previous transaction block. Since SEPAR supports both internal and cross-platform tasks and more than one platform are involved in each cross-platform transaction, similar to other works from Amiri et al. [42] [43], the ledger is formed as a *directed acyclic graph (DAG)* where the *nodes* of the graph are transaction blocks (each block includes a single transaction) and *edges* enforce the order among transaction blocks. In addition to submission, claim, and verification transactions, a unique initialization transaction (block), called the *genesis* transaction is also included in the ledger.

Fig. 3.3(a) shows a blockchain ledger created in the SEPAR model for a blockchain infrastructure consisting of four platforms p_1, p_2, p_3, and p_4. In this figure, λ is the genesis block of the blockchain, t_i's are submission transactions, t_ic_j is the j-th claim transaction of task t_i, and t_iv is the verification transaction of task t_i. In Fig. 3.3(a), t_{10}, t_{20}, t_{30}, and t_{40} are internal submission transactions of different platforms. In SEPAR, as can be seen, the internal transactions of different platforms can be appended to the ledger in parallel. $t_{10}c_1$, $t_{10}c_2$, ..., and $t_{40}c_2$ are the corresponding claim transactions. As shown, t_{10} requires 3 contributions (thus 3 claim transactions) whereas each of t_{20}, t_{30}, and t_{40} needs two contributions. $t_{10}v$, $t_{20}v$, $t_{30}v$, and $t_{40}v$ are also the verification transactions. $t_{11,21}$ is a cross-platform submission among platforms p_1 and p_2. Similarly, $t_{31,41}$ is a cross-platform submission among platforms p_3 and p_4. Here, $t_{11,21}$ needs a single contribution and $t_{31,41}$ requires two contributions. Note that the claim transactions of a cross-platform task might be initiated by different platforms and as mentioned earlier, the order of these claim transactions is important (to recognize the n first claims). Finally, $t_{22,32,42}$ is a cross-platform task among platforms p_2, p_3, and p_4 that is processed in parallel to the internal task t_{12} of platform p_1.

The introduced blockchain ledger includes all transactions of internal as well as cross-platform tasks initiated by all platforms. However, due to the data privacy requirement,

42. Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi, « CAPER: a cross-application permissioned blockchain », *in*: *Proc. of the VLDB Endow.* 12.*11* (2019), pp. 1385–1398.

43. Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi, « SharPer: Sharding Permissioned Blockchains Over Network Clusters », *in*: *arXiv preprint arXiv:1910.00765* (2019).

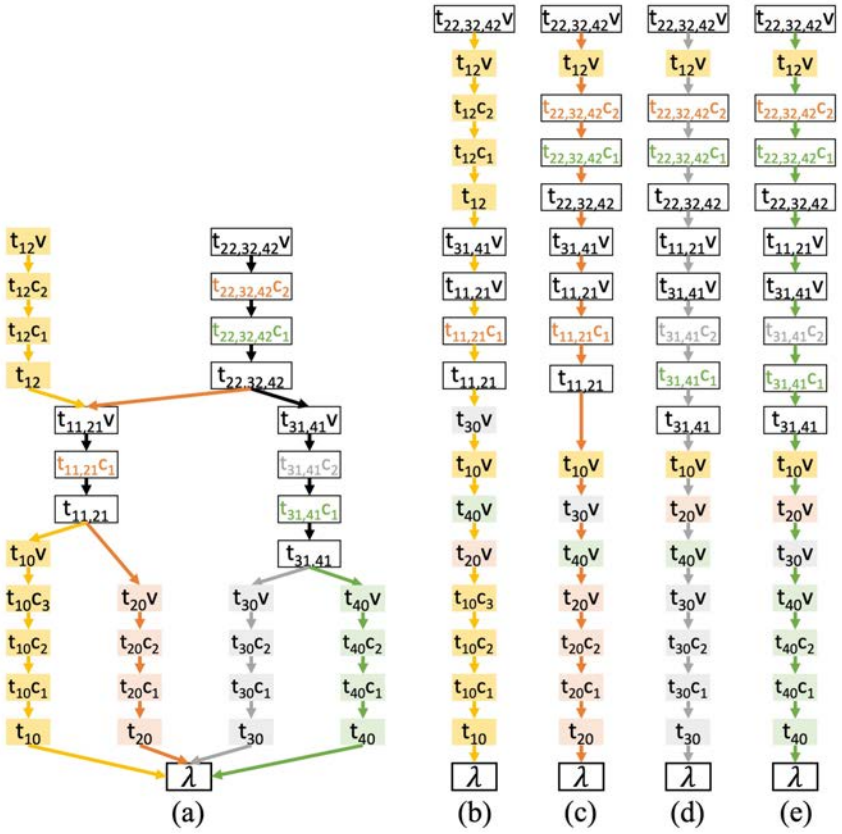


Figure 3.3 – (a): The ledger of a system consisting of four platforms, (b), (c), (d), and (e): The views of the blockchain from the four different platforms

each platform must access only a subset of these transactions, i.e., the transactions in which the platform is involved. As a result and for the sake of performance, in SEPAR, the entire blockchain ledger is *not maintained* by any platform and each platform only maintains its own *view* of the blockchain ledger including (1) all submission and claim transactions of its internal tasks, (2) all submission and claim transactions of the cross-platform tasks that the platform is involved in them, and (3) verification transactions of all tasks. Note that verification transactions are replicated on every platform to enable all platforms to check the satisfaction of constraints. The blockchain ledger is indeed the union of all these physical views.

Fig. 3.3(b)-(e) show the views of the blockchain ledger for platforms p_1, p_2, p_3, and p_4 respectively. As can be seen, each platform p_i maintains only submission and claim transactions of all internal tasks as well as cross-platform tasks that p_i is involved in them and verification transactions of all tasks. For example and as shown in Fig. 3.3(b),

platform p_1 maintains all transactions of its two internal tasks t_{10} and t_{12}. These are either submission transactions, i.e., t_{10} and t_{12}, or claim transactions, i.e., $t_{10}c_1$, $t_{10}c_2$, $t_{10}c_3$, $t_{12}c_1$, $t_{12}c_2$, or verification transactions, i.e., $t_{10}v$, $t_{12}v$. Platform p_1 also maintains cross-platform transactions that p_1 is involved in, i.e., $t_{11,21}$, $t_{11,21}c_1$, and $t_{11,21}v$. Finally, p_1 maintains the verification transactions of all other tasks within the system, i.e. $t_{20}v$, $t_{30}v$, $t_{40}v$, $t_{31,41}v$, and $t_{22,32,42}v$. Note that, since there is no data dependency between the verification transactions of the tasks that a platform is not involved in and the transactions of the tasks that a platform is involved in, the verification transactions might be appended to the ledgers in different orders, e.g., $t_{20}v$ (of platform p_2) and $t_{40}v$ (of platform p_4) are appended to the ledger of platforms p_1 and p_3 in two different orders.

3.6.2 Consensus in SEPAR

In SEPAR, each platform consists of a (disjoint) set of nodes (i.e., replicas) where the platform replicates its own view of the blockchain ledger on those nodes to achieve fault tolerance. Nodes follow either crash or Byzantine failure model. In the crash failure model, nodes operate at arbitrary speed, may fail by stopping, and may restart, however, in the Byzantine failure model, faulty nodes may exhibit arbitrary, potentially malicious, behavior. Nodes of the same or different platforms need to establish consensus on a unique order in which entries are appended to the blockchain ledger. To establish consensus among the nodes, asynchronous fault-tolerant protocols have been used. Crash fault-tolerant protocols guarantee safety in an asynchronous network using $2f+1$ nodes to overcome the simultaneous failure of any f nodes while in Byzantine fault-tolerant protocols, $3f+1$ nodes are usually needed to provide the safety property in the presence of f malicious nodes.

Completion of a crowdsourcing task, as discussed earlier, requires a single submission, one or more claim, and a verification transaction. For an internal task of a platform, submission and claim transactions are replicated only on the nodes of the platform, hence, *local consensus* among nodes of the platform on the order of the transaction is needed. For a cross-platform task, on the other hand, submission and claim transactions are replicated on every node of all (and only) involved platforms. As a result, *cross-platform consensus* among the nodes of all *involved* platforms is needed. Finally, verification transactions will be appended to the blockchain of all platforms, therefore, all nodes of *every* platform participate in a *global consensus* protocol. In this section, we show how local, cross-platform, and global consensus are established in the presence of crash-only or Byzantine nodes.

Local consensus

Processing a submission or a claim transaction of an internal task requires local consensus where nodes of a single platform, *independent* of other platforms, establish agreement on the order of the transaction. The local consensus protocol in SEPAR is pluggable and depending on the failure model of nodes, i.e., crash-only or Byzantine, a platform uses a crash fault-tolerant protocol, e.g., Paxos [44], or a Byzantine fault-tolerant protocol, e.g., PBFT [45].

The local consensus protocol is initiated by a pre-elected node of the platform, called *the primary*. When the primary p receives a valid internal transaction (either submission or claim), it initiates a local consensus algorithm by multicasting a message, e.g., accept message in Paxos or pre-prepare message in PBFT, including the requested transaction to other nodes of the platform. To provide a total order among transactions, the primary also assigns a sequence number to the request. Instead of a sequence number, the primary can also include the cryptographic hash of the previous transaction block in the message. If the transaction is a claim transaction, the primary includes the cryptographic hash of the corresponding submission transaction and any previously received claim transactions for that particular task (if any). The nodes of the platform then establish agreement on a total order of transactions using the utilized consensus protocol and append the transaction to the blockchain ledger.

Cross-platform consensus

Submission and claim transactions of a cross-platform task must be appended to the blockchains of *all* involved platforms in the same order to ensure data consistency. To process such transactions, therefore, consensus among the nodes of all (and only) involved platforms is needed. SEPAR addresses the lack of trust in the collaboration between platforms, by using an asynchronous Byzantine fault-tolerant protocol to establish consensus on the order of cross-platform transactions. Since the number of nodes of each platform depends on the utilized consensus protocol within the platform (i.e. crash fault-tolerant protocols require $2f + 1$ whereas Byzantine fault-tolerant protocols require $3f + 1$ nodes), the required number of matching replies from each platform, i.e., the quorum size, to ensure the safety of protocol depends on the failure model of nodes of the platform. We define *local-majority*

44. Leslie Lamport et al., « Paxos made simple », *in*: *ACM Sigact News* 32.*4* (2001), pp. 18–25.
45. Miguel Castro and Barbara Liskov, « Practical Byzantine fault tolerance and proactive recovery », *in*: *Proc. of TOCS'02* 20.*4* (2002), pp. 398–461.

Algorithm 1 Cross-Platform Consensus

1: **init():**
2: r := *node_id*
3: p_i := the platform that initiates the consensus
4: $\pi(p)$:= the primary node of cluster p
5: P := the set of involved platforms
6: $\pi(P)$:= the primary nodes of clusters in P
7: **upon receiving** valid transaction m and $(r == \pi(p_i))$
8: **multicast** $\langle\langle \mathsf{PREPARE}, h_i, d\rangle_{\sigma_{\pi(p_i)}}, m\rangle$ to $\pi(P)$
9: **multicast** $\langle\langle \mathsf{PROPOSE}, h_i, d\rangle_{\sigma_{\pi(p_i)}}, m\rangle$ to all nodes of p_i
10: **upon receiving** valid μ= $\langle\langle \mathsf{PREPARE}, h_i, d\rangle_{\sigma_{\pi(p_i)}}, m\rangle$ and r==$\pi(p_j)$
11: if r is not involved in any uncommitted request m' where m and m' intersect in some other platform p_k
12: **multicast** $\langle\langle \mathsf{PROPOSE}, h_j, d, r\rangle_{\sigma_{\pi(p_j)}}, \mu\rangle$ to all nodes of p_j
13: **multicast** $\langle \mathsf{ACCEPT}, h_i, h_j, d, r\rangle_{\sigma_{\pi(p_j)}}$ to P
14: **upon receiving** valid $\langle\langle \mathsf{PROPOSE}, h_i, d\rangle_{\sigma_{\pi(p_i)}}, m\rangle$ and $r \in p_i$
15: **multicast** $\langle \mathsf{ACCEPT}, h_i, d, r\rangle_{\sigma_r}$ to P
16: **upon receiving** valid $\langle\langle \mathsf{PROPOSE}, h_j, d, r\rangle_{\sigma_{\pi(p_j)}}, \mu\rangle$ and $r \in p_j$
17: **multicast** $\langle \mathsf{ACCEPT}, h_i, h_j, d, r\rangle_{\sigma_r}$ to P
18: **upon receiving** valid matching $\langle \mathsf{ACCEPT}, h_i, h_j, d, r\rangle_{\sigma_r}$ from *local-majority* of every platform p_j in P
19: **multicast** $\langle \mathsf{COMMIT}, h_i, h_j, ..., h_k, d, r\rangle_{\sigma_r}$ to P
20: **upon receiving** valid $\langle \mathsf{COMMIT}, h_i, h_j, ..., h_k, d, r\rangle_{\sigma_r}$ from *local-majority* of every platform in P
21: **append** the transaction block to the ledger

as the required number of matching replies from the nodes of a platform. For a platform with crash-only nodes, local-majority is $f + 1$ (from the total $2f + 1$ nodes), whereas for a platform with Byzantine nodes, local-majority is $2f + 1$ (from the total $3f + 1$ nodes).

SEPAR processes cross-platform transactions in four phases: prepare, propose, accept, and commit. Upon receiving a cross-platform (submission or claim) transaction, the (pre-elected) primary node of the (recipient) platform initiates the consensus protocol by multicasting a prepare message to the primary node of all involved platforms. Each primary node then assigns a sequence number to the request and multicasts a propose message to every node of its platform. During the accept and commit phases, all nodes of every involved platform communicate with each other to reach agreement on the order of the cross-platform transaction.

Algorithm 1 presents the normal case of *cross-platform consensus* in SEPAR. Although not explicitly mentioned, every sent and received message is logged by nodes. As shown in

lines 1-6 of the algorithm, p_i is the platform that initiates the transaction, $\pi(p)$ represents the primary node of platform p, P is the set of involved platforms in the transaction where $\pi(P)$ represents their current primary nodes (one node per platform).

Once the primary $\pi(p_i)$ of the initiator platform p_i receives a valid submission or claim transaction, as presented in lines 7-8, the primary node assigns sequence number h_i to the request and multicasts a *signed* prepare message $\langle\langle \textsc{prepare}, h_i, d\rangle_{\sigma_{\pi(p_i)}}, m\rangle$ to the primary nodes of all involved platforms where m is the received message (either submission or claim) and $d = D(m)$ is the digest of m. The sequence number h_i represents the correct order of the transaction block in the initiator platform p_i. If the transaction is a claim transaction, the primary includes the cryptographic hash of the corresponding submission transaction as well. As shown in line 9, the primary node also multicasts a *signed* propose message $\langle\langle \textsc{propose}, h_i, d\rangle_{\sigma_{\pi(p_i)}}, m\rangle$ to the nodes of its platform where $d = D(m)$ is the digest of m.

As indicated in lines 10-12, once the primary node of some platform p_j receives a prepare message μ from the primary node of the initiator platform, it first validates the message. If node r is currently waiting for a commit message of some cross-platform transaction m' where the involved platforms of the two requests m and m' intersect, the node does not process the new transaction m before the earlier transaction m' gets committed. This ensures that requests are committed in the same order on different platforms. Otherwise, it assigns sequence number h_j to the message and multicasts a *signed* propose message $\langle\langle \textsc{propose}, h_j, d\rangle_{\sigma_{\pi(p_j)}}, \mu\rangle$ to the nodes of its platform. The primary node $\pi(p_j)$ also piggybacks the prepare message μ to its propose message to enable the node to access the request and validate the propose message. The primary node $\pi(p_j)$, as presented in line 13, multicasts a signed accept message $\langle \textsc{accept}, h_i, h_j, d\rangle_{\sigma_{\pi(p_j)}}$ to *every* node of *all* involved platforms.

Upon receiving a propose message Once a node r of an involved platform p_j receives a propose message, as indicated in lines 8-10, it validates the signature and message digest (if the node belongs to the initiator platform ($i = j$), it also checks h_i to be valid (within a certain range)) since a malicious primary might multicast a request with an invalid sequence number. In addition, if the node is currently involved in an uncommitted cross-platform request m' where the involved platforms of two requests m and m' overlap in some other platform, the node does not process the new request m before the earlier request m' is processed. This is needed to ensure requests are committed in the same order on different platforms. The node then multicasts a signed accept message including the corresponding sequence number h_j (that represents the order of m in platform p_j), and

the digest $d = D(m)$ to *every* node of *all* involved platforms.

As presented in lines 18-19, each node waits for valid matching accept messages from a local majority (i.e., either $f + 1$ or $2f + 1$ depending on the failure model) of *every* involved platform with h_i and d that matches the propose message which was sent by primary $\pi(p_i)$. We define the predicate $\text{accepted-local}_{p_j}(m, h_i, h_j, r)$ to be true if and only if node r has received the request m, a propose for m with sequence number h_i from the initiator platform p_i and accept messages from a local majority of an involved platform p_j that match the propose message. The predicate $\text{accepted}(m, h, r)$ where $h = [h_i, h_j, ..., h_k]$ is then defined to be true on node r if and only if $\text{accepted-local}_{p_j}$ is true for *every* involved platform p_j in cross-platform request m. The order of sequence numbers in the predicate is an ascending order determined by their platform ids. The propose and accept phases of the algorithm basically guarantee that non-faulty nodes agree on a total order for the transactions. When $\text{accepted}(m, h, v, r)$ becomes true, node r multicasts a signed commit message $\langle \text{COMMIT}, h, d, r \rangle_{\sigma_r}$ to all nodes of every involved platforms.

Finally, as shown in lines 20-21, node r waits for valid matching commit messages from a local majority of *every* involved platform that matches its commit message. The predicate $\text{committed-local}_{p_j}(m, h, r)$ is defined to be true on node r if and only if $\text{accepted}(m, h, r)$ is true and node r has accepted valid matching commit messages from a local majority of platform p_j that match the propose message for cross-platform transaction m. The predicate $\text{committed}(m, h, v, r)$ is then defined to be true on node r if and only if $\text{committed-local}_{p_j}$ is true for *every* involved platform p_j in cross-platform transaction m. The committed predicate indeed shows that at least $f + 1$ nodes of each involved platform have multicast valid commit messages. When the committed predicate becomes true, the node considers the transaction as committed. If all transactions with lower sequence numbers than h_j have already been committed, the node appends a transaction block including the transaction as well as the corresponding commit message to its copy of the ledger.

In addition to the normal case operation, SEPAR has to deal with two other scenarios. First, when the primary node fails. Second, when nodes have not received a quorum of *matching* accept messages from the local-majority of every involved platform due to conflicting accept messages. Indeed, the primary nodes of different platforms might multicast their propose messages in parallel, hence, different overlapping platforms might receive the messages in different order. Furthermore, nodes might assign inconsistent sequence numbers since they have not necessarily received the latest propose message from the

Algorithm 2 Global Consensus

1: **init():**
2: $r := node_id$
3: $p_i :=$ the platform that initiates the consensus
4: $\pi(p) :=$ the primary node of cluster p

5: **upon receiving** valid transaction m and $(r == \pi(p_i))$
6: **multicast** $\langle\langle \text{PREPARE}, h_i, d\rangle_{\sigma_{\pi(p_i)}}, m\rangle$ to the primary node of every cluster
7: **multicast** $\langle\langle \text{PROPOSE}, h_i, d\rangle_{\sigma_{\pi(p_i)}}, m\rangle$ to all nodes of p_i

8: **upon receiving** valid $\mu = \langle\langle \text{PREPARE}, h_i, d\rangle_{\sigma_{\pi(p_i)}}, m\rangle$ and $r == \pi(p_j)$
9: if r is not involved in any uncommitted request m' where m and m' intersect in some other platform p_k
10: **multicast** $\langle\langle \text{PROPOSE}, h_j, d, r\rangle_{\sigma_{\pi(p_j)}}, \mu\rangle$ to all nodes of p_j
11: **multicast** $\langle \text{ACCEPT}, h_i, h_j, d, r\rangle_{\sigma_{\pi(p_j)}}$ to all nodes

12: **upon receiving** valid $\langle\langle \text{PROPOSE}, h_i, d\rangle_{\sigma_{\pi(p_i)}}, m\rangle$ and $r \in p_i$
13: **multicast** $\langle \text{ACCEPT}, h_i, d, r\rangle_{\sigma_r}$ to all nodes

14: **upon receiving** valid $\langle\langle \text{PROPOSE}, h_j, d, r\rangle_{\sigma_{\pi(p_j)}}, \mu\rangle$ and $r \in p_j$
15: **multicast** $\langle \text{ACCEPT}, h_i, h_j, d, r\rangle_{\sigma_r}$ to all nodes

16: **upon receiving** valid matching $\langle \text{ACCEPT}, h_i, h_j, d, r\rangle_{\sigma_r}$ from *local-majority* of **two-thirds** of platforms
17: **multicast** $\langle \text{COMMIT}, h_i, h_j, ..., h_k, d, r\rangle_{\sigma_r}$ to all nodes

18: **upon receiving** valid $\langle \text{COMMIT}, h_i, h_j, ..., h_k, d, r\rangle_{\sigma_r}$ from *local-majority* of **two-thirds** of platforms
19: **append** the transaction block to the ledger

primary of their own platform. We use a technique similar to SharPer [46] to address these two situations.

Global consensus

The verification transactions include group signatures and all tokens that are consumed by different participants to perform a particular task. In SEPAR and in order to enable all platforms to check constraints, verification transactions are appended to the blockchains of all platforms. To do so, a Byzantine fault-tolerant protocol is run among all nodes of

46. Amiri, Agrawal, and El Abbadi, *op. cit.*

every platform where the protocol needs agreement from the *local majority* of the nodes of *two-thirds* of the platforms. The local majority, similar to cross-platform consensus, is defined based on the utilized consensus protocol within each platform. However, there are two main differences between cross-platform consensus and global consensus. First, in cross-platform consensus only the involved platforms participate, whereas, in global consensus, every platform verifies transactions by checking the group signatures and consumed tokens. Second, cross-platform consensus requires agreement from every platform, whereas, in global consensus, agreement from only two-thirds of platforms is needed. In fact, in cross-platform consensus, there might be some dependency between cross-platform transactions and internal ones. Thus, to ensure data consistency, every involved platform must agree on the order of the cross-platform transaction. However, in global consensus, the goal is to verify the correctness of the transaction and as soon as two-thirds of platforms verify that (assuming at most one-third of platforms might behave maliciously), the transaction can be appended to the blockchain ledger.

Algorithm 2 shows the normal case of *global consensus* in SEPAR where a Byzantine protocol is run among all nodes of every platform (in contrast to cross-platform consensus where only the involved platforms participate). The protocol, similar to cross-platform consensus, process a transaction in four phases of prepare (lines 5-6), propose (lines 7-10), accept (lines 11-15), and commit (lines 16-19), however, each node waits for matching accept and commit messages from the local majority of only *two-thirds* of the platforms (as shown in lines 16 and 18).

Correctness arguments

A consensus protocol has to satisfy four main properties [47]: (1) *agreement:* every correct node must agree on the same value (Lemma 3), (2) *Validity (integrity):* if a correct node commits a value, then the value must have been proposed by some correct node (Lemma 4), (3) *Consistency (total order):* all correct nodes commit the same value in the same order (Lemma 5), and (4) *termination:* eventually every node commits some value (Lemma 6). The first three properties are known as *safety* and the termination property is known as *liveness*. In an asynchronous system, where nodes can fail, as shown by Fischer et al. [48], consensus has no solution that is both safe and live. Therefore, SEPAR guarantees safety

47. Christian Cachin, Rachid Guerraoui, and Luís Rodrigues, *Introduction to reliable and secure distributed programming*, Springer Science & Business Media, 2011.

48. Michael J Fischer, Nancy A Lynch, and Michael S Paterson, « Impossibility of distributed consensus with one faulty process », *in*: *Journal of the ACM (JACM)* 32.*2* (1985), pp. 374–382.

in an asynchronous network. However, similar to most fault-tolerant protocols, it deals with termination (liveness) only during periods of synchrony using timers.

Lemma 3. *(Agreement) If node r commits request m with sequence number h, no other correct node commits request m' ($m \neq m'$) with the same sequence number h.*

The propose and accept phases of both cross-platform and global consensus protocols guarantee that correct nodes agree on a total order of requests. Indeed, if the $\mathsf{accepted}(m, h, r)$ predicate where $h = [h_i, h_j, ..., h_k]$ is true, then $\mathsf{accepted}(m', h, q)$ is false for any non-faulty node q (including $r = q$) and any m' such that $m \neq m'$. This is true because (m, h, r) implies that $\mathsf{accepted\text{-}local}_{p_j}(m, h_i, h_j, r)$ is true for each involved platform p_j and a local majority ($f + 1$ crash-only or $2f + 1$ Byzantine node) of platform p_j have sent accept (or propose) messages for request m with sequence number h_j. As a result, for $\mathsf{accepted}(m', h, q)$ to be true, at least one non-faulty node needs to have sent two conflicting accept messages with the same sequence number but different message digest. This condition guarantees that first, a malicious primary cannot violate the safety and second, at most one of the concurrent *conflicting* transactions, i.e., transactions that overlap in at least one platform, can collect the required number of messages from each overlapping platform. Across different views, the view-change routine of SEPAR guarantees that non-faulty nodes of some platform p_j agree on the sequence number of requests that are committed-local in different views at different nodes.

Lemma 4. *(Validity) If a correct node r commits m, then m must have been proposed by some correct node π.*

In the presence of crash-only nodes, validity is ensured since crash-only nodes do not send fictitious messages. In the presence of Byzantine nodes, however, validity is guaranteed mainly based on standard cryptographic assumptions about collision-resistant hashes, encryption, and signatures which the adversary cannot subvert them. Since the request as well as all messages are signed and either the request or its digest is included in each message (to prevent changes and alterations to any part of the message), and in each step $2f + 1$ matching messages (from each Byzantine platform) are required, if a request is committed, the same request must have been proposed earlier.

Lemma 5. *(Consistency) Let P_μ denote the set of involved platforms for a request μ. For any two committed requests m and m' and any two nodes r_1 and r_2 such that $r_1 \in p_i$, $r_2 \in p_j$, and $\{p_i, p_j\} \in P_m \cap P_{m'}$, if m is committed before m' in r_1, then m is committed before m' in r_2.*

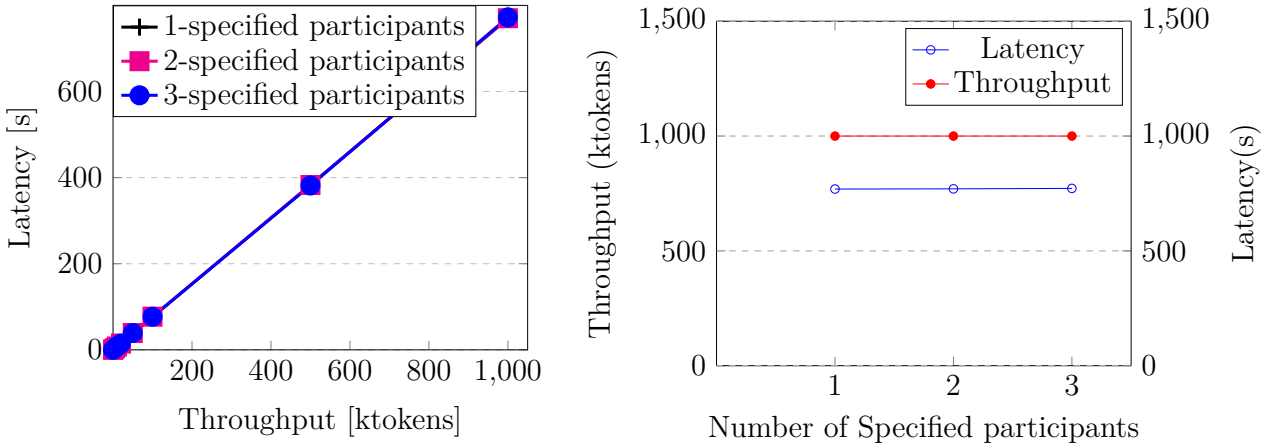


Figure 3.4 – Varying the Number of Tokens

As mentioned in both cross-platform and global consensuses, once a node r_1 of some platform p_i receives a propose message for some transaction m, if the node is involved in some other uncommitted transaction m' where m and m' overlap, node r_1 does not send an accept message for transaction m before m' gets committed. In this way, since committing request m requires accept messages from a local majority of *every* (involved) platform, m cannot be committed until m' is committed. As a result the order of committing messages is the same in all involved platforms.

Lemma 6. *(Termination) A request m issued by a correct client eventually completes.*

In an asynchronous system, where nodes can fail, as shown by Fischer et al. [49], consensus has no solution that is both safe and live. Therefore, SEPAR, similar to most fault-tolerant protocols, deals with termination (liveness) only during periods of synchrony using timers. To do so, three scenarios need to be addressed. If the primary is non-faulty and accept messages are non-conflicting, following the normal case operation of the protocol, request m completes. If the primary is non-faulty, but accept messages are conflicting, the request will be re-initiated. Finally, view change routines handle primary failures.

3.7 Experimental evaluations

In this section, we conduct several experiments to evaluate SEPAR. We have implemented a blockchain-based multi-platform crowdsourcing system. For the purpose of this

49. *Ibid.*

evaluation, and as explained earlier, we do not focus on the description of tasks and contributions (both are modeled as arbitrary bitstrings). In addition, certificate tokens, as explained earlier, are very similar to $((w, p, r), \theta)$ tokens except for the private part that has no significant impact on the performance and the number of interaction phases which is even less than constraint tokens. Therefore, we only focus on constraint tokens in the experiments. To implement group signatures, as discussed in Section 3.2, we use the protocol proposed by Camenisch et al. [50]. The experiments were conducted on the Amazon EC2 platform. Each VM is c4.2xlarge instance with 8 vCPUs and 15GB RAM, Intel Xeon E5-2666 v3 processor clocked at 3.50 GHz. When reporting throughput measurements, we use an increasing number of tasks submitted by requesters running on a single VM, until the end-to-end throughput is saturated, and state the throughput and latency just below saturation.

3.7.1 Token generation

In the first set of experiments, we measure the performance of token generation in SEPAR for different types of constraints. We consider constraints with a single specified participant (e.g., $((w, *, *), \theta)$), two specified participants (e.g., $((*, p, r), \theta)$), and three specified participants (e.g., $((w, p, r), \theta)$). As shown in Figure 3.4(a), SEPAR is able to generate tokens in linear time. SEPAR generates each token in $0.7ms$, hence, generating 1 million tokens in 12 minutes. This is an acceptable amount of time since token generation is executed periodically, e.g., every week or every month. Note that since tokens of different constraints can be generated in parallel, SEPAR can easily parallelize the token generation routine in order to improve the throughput. As can be seen in Figure 3.4(b), the type of constraints, i.e., the number of specified participants, also does not affect the performance and the token generation throughput and latency is constant in terms of the number of participant. However, it should be noted that a more complicated constraint, i.e., a constraint with more specified participants, requires more tokens to be generated.

3.7.2 Impact of cross-platform tasks

In the second set of experiments, we measure the performance of SEPAR for workloads with different percentages of cross-platform tasks. We consider four different workloads with (1) no cross-platform tasks, (2) 20% cross-platform tasks, (3) 80% cross-platform tasks, and

50. Camenisch and Groth, *op. cit.*

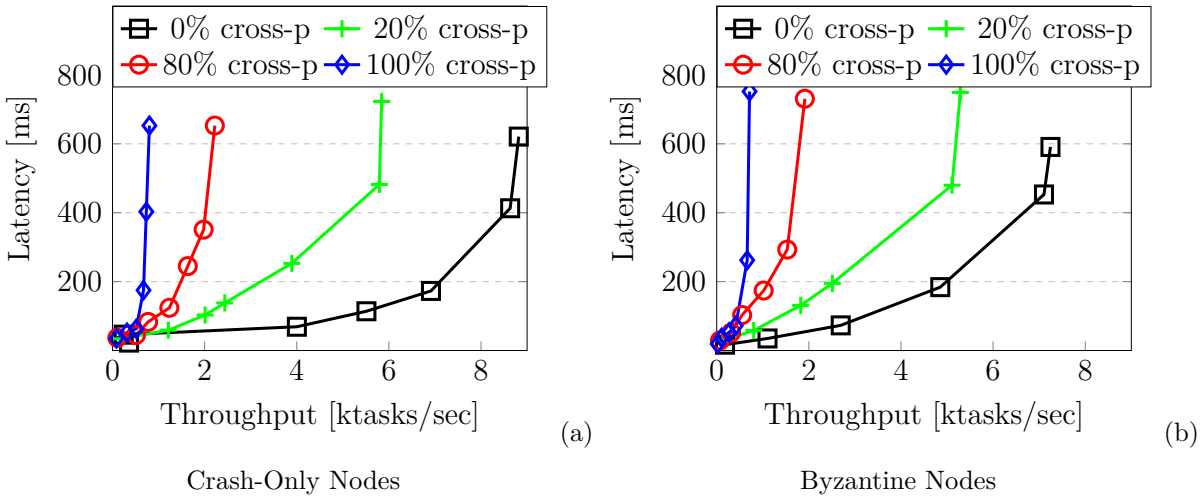


Figure 3.5 – Varying Number of Cross-Platform Tasks

(4) 100% cross-platform tasks. We also assume that two (randomly chosen) platforms are involved in each cross-platform tasks and completion of each task requires a contribution coming from a randomly chosen worker. The system includes four platforms and each task has to satisfy two randomly chosen constraints. We consider two different networks with crash-only and Byzantine nodes. When all nodes follow crash-only nodes, as presented in Figure 3.5(a), SEPAR is able to process 8600 tasks with 400 ms latency before the end-to-end throughput is saturated (the penultimate point), if all tasks are local. Note that even when all tasks are local, the verification transaction of each task still needs global consensus among all platforms. Increasing the percentage of cross-platform tasks to 20%, reduces the overall throughput to 5800 (67%) with 400 ms latency since processing cross-platform tasks requires cross-platform consensus. By increasing the percentage of cross-platform tasks to 80% and then 100%, the throughput of SEPAR will reduce to 1900 and 700 with the same (400 ms) latency. This is expected because when most tasks are cross-platform ones, more nodes are involved in processing a task and more messages are exchanged. In addition, the possibility of parallel processing of tasks will be significantly reduced. In the presence of Byzantine nodes, as shown in Figure 3.5(b), SEPAR demonstrates the similar behavior as the previous case (crash-only nodes). When all tasks are local, SEPAR processes 7100 tasks with 450 ms latency. Increasing the percentage of cross-platform tasks to 20% and 80% will reduce the throughput to 4900 and 1700 tasks with the same (450 ms) latency respectively. Finally, when all tasks are cross-platform, SEPAR is able to process 700 tasks with 450 ms latency.

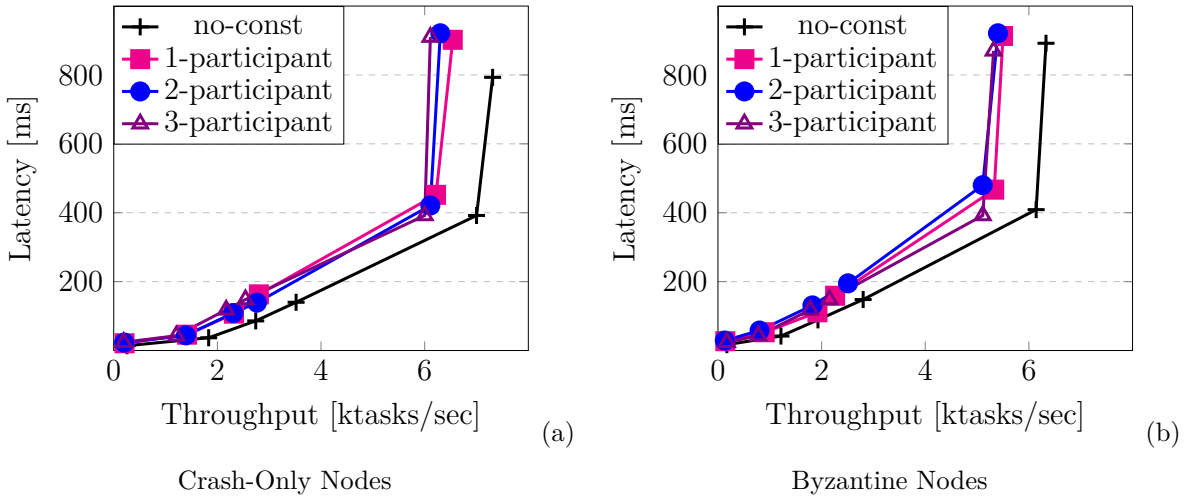


Figure 3.6 – Varying the Type of Constraints

3.7.3 Varying the types of constraints

In the next set of experiments, we measure the performance of SEPAR with different types of constraints. We consider four different scenarios where each task has to satisfy (1) no constraints (i.e., basic scenario), (2) a one-specified constraint, (3) a two-specified constraint, and (4) a three-specified constraint. The system consists of four platforms and the workload includes 90% intra- and 10% cross-platform tasks (the typical settings in partitioned databases [51] [52]) where two (randomly chosen) platforms are involved in each cross-platform tasks. As before, completion of each task requires a single contribution. To measure the overhead of group signatures and tokens, we compare the results with the basic scenario where there is no constraints in the system, thus, there is no need to exchange and validate tokens and signatures. When nodes follow the crash failure model and the system has no constraints, as can be seen in Figure 3.6(a), SEPAR is able to process 7000 tasks with 390 ms latency before the end-to-end throughput is saturated (the penultimate point). Adding constraints to the tasks results in more phases of communication between different participants to exchange tokens and signatures, however, SEPAR is still able to process 6200 tasks (the penultimate point) with 450 ms latency (only 11% and 15% overhead in terms of the throughput and latency respectively). The number of participants in each constraint, on the other hand, does not significantly affect the performance of SEPAR. This is expected,

51. Alexander Thomson, Thaddeus Diamond, et al., « Calvin: fast distributed transactions for partitioned database systems », *in*: *Proc. of MOD'12*, ACM, 2012, pp. 1–12.

52. Rebecca Taft, Essam Mansour, et al., « E-store: Fine-grained elastic partitioning for distributed transaction processing systems », *in*: *Proc. of the VLDB Endow.* 8.*3* (2014), pp. 245–256.

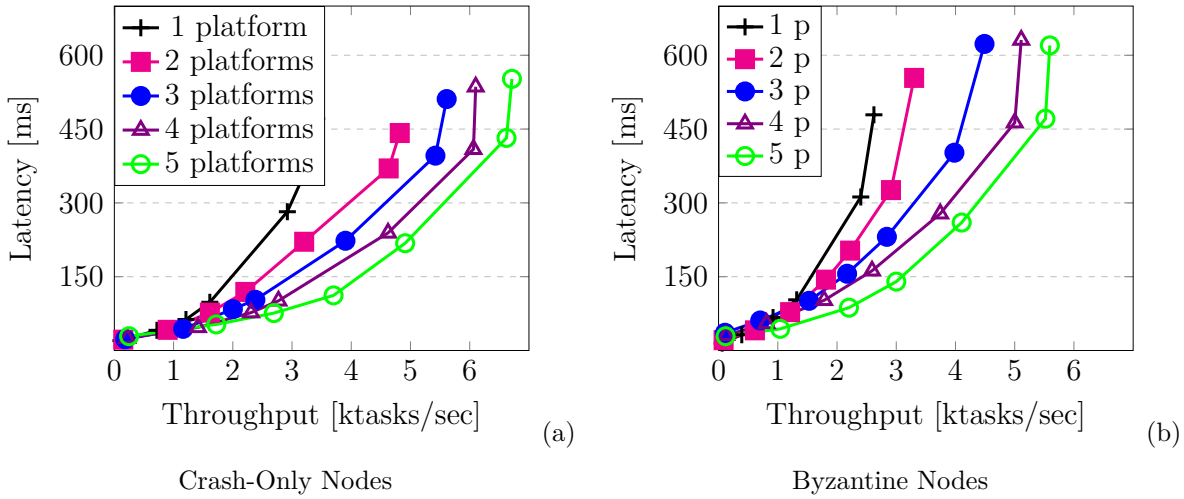


Figure 3.7 – Varying the Number of Platforms

because more participants results in only increasing the number of (parallel) tokens and signature exchanges and the consensus protocols and other communication phases are not affected. Similarly, in the presence of Byzantine nodes and as shown in Figure 3.6(b), SEPAR is able to process 6140 tasks with 409 ms latency with no constraints and 5331 tasks (13% overhead) with 467 ms (14% overhead) latency with one-specified constraint. As before, the number of participants does not significantly affect the performance.

It should be noted that with an increased number of constraint, SEPAR still demonstrates similar performance as shown in this experiment (increasing the number of participants in each constraints). Indeed, adding more constraints results in adding more tokens and possibly more participants and signatures, however, it does not affect the consensus protocols and other communication phases.

3.7.4 Varying the number of platforms

In the last set of experiments, we measure the scalability of SEPAR in crowdsourcing systems with different number of platforms. We measure the performance of SEPAR in networks including 1 to 5 platforms for both crash-only and Byzantine nodes (assuming $f = 1$ in each platform). Each task has to satisfy on average two randomly chosen constraints, two (randomly chosen) platforms are involved in each cross-platform tasks, completion of each task requires a single contribution, and the workloads include 90% intra- and 10% cross-platform tasks. Note that in the scenario with a single platform, all tasks are intra-platform. As shown in Figure 3.7(a), in the presence of crash-only nodes,

the performance of the system improves by adding more platforms, e.g., with five platform, SEPAR processes 6600 tasks with 400 ms latency whereas in a single platform setting, SEPAR processes 3300 task with the same latency. While adding more platforms improves the performance of SEPAR, the relation between the increased number of platforms and the improved throughput is non-linear (the number of platforms has been increased 5 times while the throughput doubled). This is expected because adding more platforms while increasing the possibility of parallel processing of local tasks, makes the global consensus algorithm (which is needed for every single task) more expensive. In the presence of Byzantine nodes, SEPAR demonstrates similar behavior, e.g., processes 5500 tasks with 470 ms latency with 5 platforms.

3.8 Conclusion

In this chapter, we introduce SEPAR, a multi-platform crowdsourcing system that enforces global regulations in a privacy-preserving and transparent manner. SEPAR consists of two main components. First, a token-based system that enables official institutions to express legal regulations in simple and unambiguous terms, guarantees the satisfaction of global constraints by construction, and allows participants to prove to external entities their involvement in crowdsourcing tasks, all in a privacy-preserving manner. Second, a permissioned blockchain that provides transparency using distributed ledgers shared across multiple platforms and enables collaboration among platforms through a suite of distributed consensus protocols. To the best of our knowledge, SEPAR is the first to address the problem of enforcing global regulation over multi-crowdsourcing platforms. We prove the privacy requirements of the token-based system as well as the correctness of the consensus protocols and conduct an extensive experimental evaluation to measure the performance and scalability of SEPAR.

Chapter 4

Towards extending PIR to multiple downloads

4.1 Introduction

Private Information Retrieval, or PIR for short, is a family of techniques that allows clients to download an item from a list stored on a server, without the server knowing which item is downloaded. In the context of crowdsourcing, PIR allows workers to download tasks without the platform knowing what they are interested in, as developed in Chapter 2. However, the security model of PIR, introduced by Chor et al. [1], only states security properties for *individual* queries, while applications often use it for multiple queries, such as episodes of series [2].

This focus on individual queries can be seen as a limitation when it comes to security, and the claims of PIR can be misleading when using multiple queries, making clients, platforms, or even developers think PIR provides a better security than it actually does. However, to the best of our knowledge, no related works ensure that multiple uses of PIR preserve the security properties of a single query. In order to illustrate the reasons for which considering this omission is an issue, we propose the three following examples:

Episodes of series Let consider a case in which items to be downloaded are closely interconnected, such as episodes of series, in a media provider context (*e.g.* Netflix), providing PIR to protect its clients [3].

Let consider three series: one of 12 episodes, one of 14 episodes, and one of 15 episodes and a client who downloads 14 items. Even without having access to a large study on the behaviour of clients, it is easy to see that all episodes of all series do not hold the same

1. Chor, Goldreich, et al., *op. cit.*
2. Gupta, Crooks, et al., *op. cit.*
3. *Ibid.*

probability of having been downloaded As people usually watch series untill the end, the client probably downloaded the 14 episodes series. Even if it was not the case, watching the last episode of a series while missing one episode is very unusual, so the 15th episode of the longest series was probably not included in the downloads.

Statistical knowledge on population In a similar way, considering a movie database, we could also consider external knowledge on the population. Let say for instance that people watching a lot of movies mostly look at old movies. Then, it is immediate to see that knowing the number of downloads performed by a client leaks statistical information on the downloaded content.

Crowdsourcing In crowdsourcing, workers perform tasks that are provided by requesters, using a platform as an intermediary. In the context of topic-aware tasks and workers (meaning that workers have some degree of skill in some topics and that tasks have some requirements on these skills, similar to the model used in Chapter 2), workers may want to download tasks without the platform knowing what they are interested in, and the platform can implement a PIR protocol to provide this guarantee.

In such a situation, knowing the requirements of tasks (*e.g.* a task that can be given only to workers who are good in software development, no matter their skills in cooking) and how many tasks a worker downloads (*e.g.* a worker downloads 50 tasks), some information can be deduced on which tasks are downloaded, although this is the information PIR is supposed to protect. For instance, if we know that the requirements of the 50 downloaded tasks intersect in at least one point, this knowledge may exclude some possibilities (*e.g.* if a worker downloads 50 tasks, and 50 tasks intersect only with high requirements in software development skills, we can deduce that the worker is a good software developer).

The main issue that is highlighted in the above examples is that queries are rarely completely independent from each other. As a consequence, the PIR definition ensuring that each individual query is protected cannot be extended to multiple queries without any change. Therefore it seems necessary to extend this security of PIR to take into account sets of queries. As a privacy requirement, we propose here to extend the original and informal formulation of Chor et al. [4] to match multiple queries and desired items in the following: our privacy requirement is that each **set of queries** is distributed independently

4. Chor, Goldreich, et al., *op. cit.*

of the set of desired items, so that the server gains no information about the interests of the user.

Note that the set of queries does not necessarily contain as many items as the set of desired items (*e.g.* a query for 2 items has to be indistinguishable from a query for 50 items). Although this informal definition is not suitable for formal proofs of security, we believe that the diversity of models and assumptions used in PIR (from information theory to cryptography) makes it hard, if not impossible, to propose a formal unified model. In order to prove that algorithms respect privacy models, it is assumed that more formal models will be built according to the specificities of protocols (*e.g.* assuming non-colluding parties, or limits on computation power).

Also, note that Chapter 2 takes these limitations into account when PIR techniques are used to match tasks and workers: the current chapter is an attempt to extend the scope of the previous work, in order to better understand the limitations of PIR, and the ways to overcome them.

However, this chapter is a preliminary work and we do not claim our analysis to be exhaustive, nor that our attack is the only one possible. We mostly aim at emphasizing possible side-channel attacks, and at opening discussions on this important matter: both experiments and in-depth security analysis are future works to be done. Yet, we believe that the work provided in this section provides interesting material to consider limitations of PIR, and build counter-measures that avoid them in future work.

In this chapter, we propose the following contributions:

— A model describing external knowledge on a dataset, along with an attack model exploiting this knowledge together with naive use of PIR revealing the number of downloads.
— An attack, following this attack model, that manages to retrieve information illegitimately on the downloaded items.
— Instantiations of this attack with the three examples presented above.
— Possible counter-measures in order to (1) respect naive instantiations of our informal privacy requirement for multiple queries, or (2) mitigate the attack if the cost of privacy on efficiency were to be too heavy.

In Section 4.2, we provide the necessary background on PIR protocols. In Section 4.3, we develop an attack model, together with an attack, and explain how this model fits our examples. We propose in Section 4.4 a short review of possible counter-measures that we envision, either to match the security of PIR techniques with multiple downloads, or at

least to mitigate our attack. Possible future works are presented in Section 4.5, and we conclude in Section 4.6.

4.2 Related Work

In this section, we present the two main families of Private Information Retrieval techniques. A Private Information Retrieval (PIR) protocol [5] is a protocol that makes it possible for a user to download an *item* (*e.g.* a record, a movie, etc.) from a *library* (an ordered list of items) from a server or from a set of servers, without the server(s) knowing which item of the list has been downloaded. As stated by Chor et al. [6]: "The privacy requirement is that each individual query is distributed independently of [the desired item] and thus the server gains no information about the identity of the desired item."

A naive approach to this problem is to allow the user to download the whole library, as the servers would be unable to learn which item the user was interested in. However, this approach requires a significant overhead in bandwidth. Two main approaches exist to PIR protocols: computational PIR (cPIR) and Information Theoretical PIR (ITPIR). Although both families have several variations and implementations, we explain here their basic requirements and main ideas. It is also noticeable that some approaches propose to use both, in order to get most of each method [7] [8].

4.2.1 Computational PIR

Computational PIR makes use of cryptographic tools to create PIR protocols. As a result, it also inherits from cryptographic security hypothesis on computation time and power of attackers.

We present here a simplified version of the protocol proposed by Kushilevitz et al. [9], as an example. This protocol relies on homomorphic encryption schemes, providing (1) addition (we denote $+_h$ - respectively Σ_h - the operation that inputs two encrypted values - respectively any number of them - and outputs the encrypted sum of their clear values)

5. *Ibid.*
6. *Ibid.*
7. Gupta, Crooks, et al., *op. cit.*
8. Casey Devet and Ian Goldberg, « The best of both worlds: Combining information-theoretic and computational PIR for communication efficiency », *in*: *Proc. of PETS'14*, 2014, pp. 63–82.
9. Eyal Kushilevitz and Rafail Ostrovsky, « Replication is not needed: Single database, computationally-private information retrieval », *in*: *Proc. of FOCS'97*, IEEE, 1997, pp. 364–373.

and (2) scalar multiplications (we denote $\times_h$ the operation that inputs an encrypted value and a clear value, and outputs the encrypted product of the corresponding clear values of both inputs). We consider a library L of n binary objects (items). It is assumed that all items share the same length in bits, denoted l, that each item has a unique identifier, and that users know the list of ids of the existing items in L. The library is stored as a vector of l-bit integers: $L \in (\{0,1\}^l)^n$. Now a client wants to retrieve the item of id i. First, she instanciates a vector of n bits, initializes all bits to 0s, and sets to 1 the bit at position i. She then encrypts each bit separately, and sends the resulting encrypted vector - denoted c - to the server. After that, the server computes $r = \sum_{h,i=1}^{n} c[i] \times_h L_i$ and sends it back to the client. Actually, r is a sum of (1) encrypted 0s (corresponding to the encrypted 0s in c) and (2) an encrypted bit-subsequence of the requested binary object (corresponding to the encrypted 1 in c), which are undinstinguishable for the server. Third and finally, the user decrypts r, obtaining the requested item.

Although this protocol diminishes communication costs in most cases compared to the naive version (sending everything), the computation cost increases a lot (for instance, servers have to read the whole dataset, and to perform computations on it). For this reason, cPIR has been mostly disregarded. For example, an analysis provided by Sion et al. [10] argued that the use of cPIR in a single-server context would always be slower than the naive PIR protocol (according to the evolution of hardware and bandwidth, computation costs were to exceed benefits from lower requirements in communication). However, a recent line of work initiated by Aguilar-Melchor et al. [11] disputes these claims by making use of more efficient cryptographic schemes, thus making cPIR significantly faster.

4.2.2 Information theoretical PIR

Information Theoretical PIR does not assume limitations on the computation time or power of the attacker. However, other assumptions are used on the behaviours of servers (*e.g.* hypothesis on collusions between servers, on the number of servers giving incorrect answers, etc.). We explain here one of the many existing protocols, introduced by Chor et al. [12], as an example.

This protocol relies on the XOR operator, which takes as inputs two bits, and outputs

10. Radu Sion and Bogdan Carbunar, « On the computational practicality of private information retrieval », *in*: *Proc. of NDSS'07*, 2007, pp. 2006–06.
11. Aguilar-Melchor, Barrier, et al., *op. cit.*
12. Chor, Goldreich, et al., *op. cit.*

1 if and only if they are different. When used on multiple bits, it outputs 1 if and only if the number of 1-bits is uneven (which is the same as a bitwise sum). We first assume, just as for cPIR protocols, that all items share the same length l in bits, that each item has a unique id between 0 and $n-1$, and that users know the list of ids of the existing items in L. Furthermore, we denote k the number of servers, and assume that not all servers collude together. If a user requests an item i, she will first build a bitstring of size n. This bitstring, called e_b, is composed of 0 at all positions, except for $e_b[i]$ which is equal to 1. Then, the user generates $k-1$ bitstrings of n random bits (called q_j, j between 0 and $k-2$). A last bitstring of similar length, is computed, for each bit, as the XOR of the corresponding bit of all q_j and e_b. This last vector is q_{k-1}. Note that $XOR_{j=0}^{k-1} q_j = e_b$, and that knowing all but one q_j is not sufficient to deduce the real query e_b (which explains the hypothesis on collusions). Each of these q_j is sent to a different server. At reception, the server will compute the matricial product $r_j = q_j \times L$, and send the result to the user. Finally, the user only has to compute a XOR of all results to obtain the requested item.

These mechanisms, relying on distribution and information theory, are interesting as they are faster than computational PIR (for instance XOR computations are very fast) and avoid hypothesis on the power of the attacker. However, in many real-life applications, non-collusion assumptions on servers can be hard to reach.

4.3 Attacks

In order to be more concrete about the dangers of naive uses of PIR, we propose here an attack against it. This attack considers a PIR use that protects neither the identity of users (their queries are traceable) nor the number of queries. Furthermore, external knowledge on the stored items is also accessible by the attacker. We first develop the knowledge of the attacker, and then some possible attacks. Note that neither the attack model nor the attack itself are meant to reflect exhaustively the possible weaknesses of PIR: we only attend to illustrate possible dangers.

4.3.1 Model

In order to take into account the examples presented in the introduction, we propose to model the knowledge of an attacker with two different pieces of information. First, information on the client: the provider or the attacker knows how many items the client

downloads. Then, information on the database: this includes relationship between items, and how frequently they are downloaded.

Information on the client

We here assume that the client uses a PIR protocol respecting the privacy requirement stated by Chor et al. [13], for a single query, but no more: "The privacy requirement is that each individual query is distributed independently of [the desired item] and thus the server gains no information about the identity of the desired item."

However, as the requirement for multiple queries is not respected, we consider that the attacker has access to the number N of queries of the client.

External knowledge

To model the information on the database, which is independent from the targeted client, we consider the following notations. Let D denote the database of downloadable items, $|D|$ its size, and $i_1 \dots i_n$ the items it contains, where $n = |D|$. We also consider an external knowledge on a population P: for simplicity, P is here instantiated as a database of past downloads of a representative population. The external knowledge accessible by attackers on P is instantiated with a count function C, that inputs a set of items and outputs the number of people who downloaded it. This knowledge is used as statistical information on the behaviour of clients. From this knowledge, we deduce what we call *relational knowledge* denoted $K_{D,P}$. This function takes as input a set S of items from D, and its output $K_{D,P}(S)$ is the probability that this set of items is the one that interests a client who downloaded $|S|$ items. More precise definitions are provided in Definition 13.

Definition 13 (Relational knowledge)**.** Consider a dataset D, composed of items $i_k, k \in 1, n$, and a population P, composed of individuals who all downloaded a set of items included in D. We denote $C(S)$ the function that equals the number of users who downloaded the set of items $S \in D$. The relational knowledge on D regarding P, denoted $K_{D,P}$, is a function which takes as input a set S of items of D, and outputs the proportion of users of P who downloaded exactly S among users who downloaded the same number of items:

$$K_{D,P}(S) = \frac{C(S)}{\sum_{S' \in D, |S'|=|S|} C(S')}$$

13. *Ibid.*

Note that, with this definition, the sum of the relational knowledge on all sets of a given size always equals 1.

With these notations, it is possible to model the information on the database with a weighted hypergraph. An hypergraph is a generalization of a graph, in which an edge is defined as the set of vertices that it connects together, instead of a couple of them. In our context, we consider a weighted hypergraph, meaning that edges also have a weight: they are weighted sets of vertices. For an edge E, we denote $w(E)$ its weight, and $s(E)$ its set of vertices. The hypergraph is designed as follows. Each item of the downloadable library is represented as a vertex. An edge E is a weighted set of vertices, meaning that the set of all edges is the powerset of vertices. The weight of an edge E is then $w(E) = K_{D,P}(s(E))$, the probability that this set of item is downloaded by a client who downloads exactly $|s(E)|$ items. For simplicity, we consider that an edge of null weight doesn't belong to the hypergraph.

An illustration is provided in Figure 4.1, with three items. In this example, we consider that items are movies, and that movie i_3 is the direct following of movie i_1, and that both are comedies. Movie i_2 is an horror movie. As a result, movie i_3 is never downloaded if i_1 is not. Furthermore, the movie i_2 is more often downloaded alone than any other movie, although it is more likely that i_1 is downloaded with i_3 than with i_2.

More formally, we propose the definition 14.

Definition 14 (Hypergraph of knowledge)**.** An hypergraph of knowledge H, related with a population P and a dataset D is a weighted hypergraph designed as follows:

— There is exactly one vertex for each item in D. Vertex v_k corresponds to item i_k, and they will be used equivalently in the following.

— An edge is a weighted set of item. The weight $w(E)$ of an edge E is the relational knowledge of the set of items $s(E)$ of E: $w(E) = K_{D,P}(s(E))$. If this value equals 0, the edge is not represented in the hypergraph.

Examples

Although information in this model may be hard to obtain in real life, it seems in fact quite easy to access at least partial information about it thanks to simple heuristics. To make it more concrete, we instantiate this definition in the example provided in the introduction.

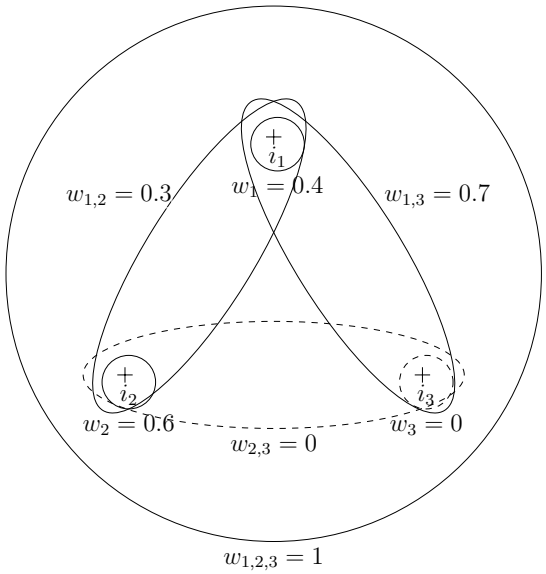


Figure 4.1 – Representation of an hypergraph of knowledge: crosses are items (vertices); ellipses and circles are sets of items (edges); w are the relational knowledge about corresponding sets of items: *e.g.* $w_{a,b,c} = K_{D,P}(\{i_a, i_b, i_c\})$; dashed lines represent null probability of download ($K_{D,P}(s(E)) = 0$)

Episodes of series In the case of series, vertices would be episodes, and edges are weighted sets of episodes, from any series. Many edges, of many sizes are likely to have a non-zero weight, but heuristics seems easy to create, and quite intuitive. For instance, it is unlikely that an episode of a series is downloaded if previous episodes are not. Therefore, edges containing such episodes are likely to have a very small weight. Similarly, if all series have the same number of episodes, downloading a multiple of this number likely means downloading complete series, leading to heavier weights on corresponding edges than on other edges of same cardinality.

Statistical knowledge on population For a movie database, each vertex would be a movie, and edges would be weighted sets of movies. With statistical knowledge, such as "the more movies someone watches, the more likely it is that these movies are old ones", it is possible to infer that large edges containing old movies are likely to be heavier that large edges that do not (and reciprocally for smaller edges).

Crowdsourcing For a crowdsourcing database, vertices would represent tasks, and edges weighted sets of tasks. In this example, requirements of the tasks will have a huge impact on the structure of the hypergraph. Indeed, tasks whose requirements do not

intersect will not be downloaded by a single client, as the client cannot match both requirements. Therefore, any edge containing tasks with non-intersecting requirements will weight 0 and not be represented. Weights for other sets of tasks may be estimated through the provided reward or the similarity between tasks and their requirements. In other words, the overall structure can be a lot lighter, and the maximum size of an edge is likely to be rather small (compared to the previous examples).

4.3.2 Attack

With this very rich model, several attacks are possible, depending on what the attacker wants to access. We propose here a naive attack with our model.

First, we consider an attacker who wants to know whether a set of items (called *target set*) has been downloaded or not by a *target client*. Note that the target client may download more items than included in the target set. With our model, the attack is quite simple: it is enough to consider all edges of cardinality equal to the number of downloads of the target client. Among this set of edges, we can compute the probability of the target set being downloaded by the target client as the sum of the weights of all edges containing the target set of items. Note, with this attack, the following cases:

— If the cardinality of the target set is larger than the number of downloads of the client, the probability that it is downloaded is null.

— If it is the same, the probability is equal to the weight of the edge containing exactly the set: $K_{D,P}(t)$, for t the target set.

Then, in a very similar way, if the attacker doesn't know what she is looking for, it is possible to compute a list of item sets, ordered by their probability of download. To do so, it is enough to take a size of set as input, and to compute, for all sets of items of this size, the probability it is downloaded. A sorted list of these probabilities is provided as an output.

Algorithm 4 sums up both aspects of this attack.

4.4 Counter-measures

We here consider various counter-measures. First, we extend our definition of packing, already used in Chapter 2. Despite its bandwidth overhead, we believe that it is the closest one to a direct extension of PIR protocols, in the way that it strictly fulfills our multi-

Algorithm 4: Attack algorithm

Data:
N: Number of queries of the client
D: Dataset of items
P: Representative population
t: Target set of items
Result:
List: List of all possible downloaded sets of items, ordered from the most probable to the least probable
P_t: The probability that all items within t have been downloaded

```
1  P_t = 0
   /* P_t is initialized at 0                                          */
2  List = []
   /* List will be filled with possible sets                           */
3  for all sets S_i of items within D, of size N
   /* There are (|D| choose N)                                         */
4  do
5      proba = K_{D,P}(S_i)
       /* Compute the probability that S_i is the downloaded set of items
          */
6      if t ⊂ S_i then
7          P_t = P_t + proba
8      List = List + (S_i, proba)
9  Sort List in descending order, according to their second element (probability of
     being downloaded)
10 return List, P_t
```

query security requirement, without requiring complex architectures (in the number of involved parties) nor additional properties (such as unlinkability between requests). Then, we consider other defenses, that rely on additional mechanisms and provide interesting protections to mitigate the attack studied in Section 4.3. However, these counter-measures seem to be inadequate as a PIR successor, either because they do not perfectly fit our additional requirement, or because they require further assumptions that are optional in a single-request PIR context.

This section presents interesting ways to counter our attack. However, proofs and experiments of these counter-measures are not provided, and are to be looked at carefully in future work.

4.4.1 Packing

A first way to protect clients against attackers is to make their behaviour indistinguishable from each other. This can be achieved by making all of them download the same number of items, even if they do not need that many. This method has already been proposed in the context of crowdsourcing in Chapter 2, and we propose here to generalize it.

We assume that there exists a maximum limit on the number of downloads and call h this limit. This limit may come from a server or economic limitation (*e.g.* only h downloads are allowed), from empirical observations (*e.g.* no edge contains more than h items), or any other reason. By definition, no client will download more than h items, and therefore, if all of them download h items (or cannot be distinguished with someone that does), an attacker will not be able to know how many items their target really intended to download.

To allow this, we propose to use PIR on what we call *packing* of items instead of isolated items, in a similar way as Definition 7 in Chapter 2.

Definition 15 (Packing, Bucket)**.** We consider a dataset D of items i_k. A packing L is a PIR library that fulfills the following conditions. In order to avoir any confusion with items, we call *buckets* the objects in such a PIR library (that can be downloaded through PIR).

1. **Security condition** Each client downloads the same number of buckets. This number is set to 1.

2. **PIR requirement** Each bucket has the same size in bits (padding is allowed):

$$\forall b_1, b_2 \in L, ||b_1|| = ||b_2||$$

   This condition comes from the use of PIR.

3. **Availability condition** For all sets of items S for which $K_{D,P}(S) \neq 0$, there has to be at least one bucket containing all items in S.

This method requires to create buckets for all possible sets of items which can be required (*e.g.* for all edges of the hypergraph). However, it is noticeable that a single bucket can be used for various sets of items (*e.g.* a bucket containing items a and b can be used for requests $(a), (b)$ or (a, b)). It is noteworthy that all buckets do not necessarily contain h items or more: for instance, if a single item is very heavy, but those who download it never download any other items, it is possible to make it a single-item bucket (with padding if necessary).

This protocol does not assume strong hypothesis, and still allows for PIR privacy properties. However, this does not come without any cost. Indeed, if h is too large (in the worst case, the size of the database itself), clients will endure huge download overcosts, up to the whole database. Optimizations and heuristics are possible, as proposed in Chapter 2, or even combined with recursive versions of PIR [14], but are context specific and thus out of our focus. Therefore, this protocol is likely not suitable if h is large, but may work well when the largest number of donwload is small, for instance in a topic-aware crowdsourcing context.

4.4.2 Mitigating the attack

In this section, we provide other possible counter-measures that do not perfectly match the PIR requirement, either because they lack guarantees, or because they add requirements. We see these counter-measures as interesting tracks to protect against the attack presented in Section 4.3, in cases where packing is not possible. Although this attack is only one among many others, we believe protecting against it is a first step towards complete and sound protections.

14. Julien P Stern, « A new and efficient all-or-nothing disclosure of secrets protocol », *in*: *Proc. of ASIACRYPT'98*, Springer, 1998, pp. 357–371.

Non-colluding parties

In order to exploit the attack developed in Section 4.3, an ill-intended party must gather two things: first, the knowledge graph, and then, the number of downloads of the client. A first naive counter-measure consists in splitting this knowledge, to ensure no-one has access to both pieces of information.

In order to do so, we propose a distributed protocol inspired from the one used by Gupta et al. [15], which uses independent parties to achieve efficiency without revealing data. We here assume the following hypothesis:

- The provider does not require to know the amount of items accessed by the client (*e.g* it provides an access through monthly subscription)
- There exist at least two third parties that:
  - do not collude with the provider
  - do not collude with each other
  - do not have access to any external information on the library (this includes, for instance, the knowledge that the database contains series, which may be a hard criteria to reach if the client is Netflix)

Although these hypothesis are hard to achieve in real-life scenarios, we believe that it is interesting to see how they can articulate to provide security guarantees. The idea behind this is to consider three servers, or sets of servers. The first one, which belongs to the provider, will grant authorization keys (*e.g.* a group signature key) to the client. The second one, a non-colluding third party, will receive the encrypted items of the library from the provider. This server is not supposed to know anything about the items. It will then provide access to these encrypted items through a PIR protocol to authorized clients. The third server will receive the decryption keys corresponding to encrypted items provided to the second server. Similarly, it does not know anything about the items, and will provide access to these keys to authorized clients, through PIR. Finally, after having received the authorization from the provider, and downloaded both the encrypted item she is interested in and the corresponding key from non-colluding servers, the client will simply decrypt the item to access it. The overall approach is summed up in Algorithm 5.

Thanks to this architecture, only the company's server knows what data are hosted, but it has no access to the number of downloads. The other two servers have access to the number of downloads, but not to the knowledge hypergraph.

15. Gupta, Crooks, et al., *op. cit.*

Algorithm 5: Non-collusion based counter-measure

Data:
L: The library of items (items are denoted $i_k, k \in [1, N]$)
P: The data provider, which decides who can access items in L
TP_1 and TP_2: Two third parties, which do not collude with each other, nor with P. They are able to perform PIR protocols
c: The client, willing to retrieve privately a list of items l
Result:
c privately retrieves all items from l
`/* Initialization:` `*/`
1 *Provider:* **for** *all items* i_k *in the library* L **do**
2 Generates a key key_k
3 Encrypts i_k with key_k, resulting in $Enc(key_k, i_k)$
`/* The choice of the encryption scheme is free` `*/`
4 Sends $Enc(key_k, i_k)$ to TP_1
5 Sends key_k to TP_2
6 Sends an authorization access to c
`/* Again, implementation details are free` `*/`
`/* Retrieving:` `*/`
7 *Client:* **for** *all items* i_k *in* l **do**
8 Gets authorization access from P to TP_1 and TP_2
9 Sends a PIR request to TP_1 to retrieve $Enc(key_k, i_k)$
10 Sends a PIR request to TP_2 to retrieve key_k
11 Decrypts $Enc(key_k, i_k)$ with key_k

Preserving Privacy

Another solution to preserve the security requirement on PIR for multiple requests is to enforce privacy on clients themselves. Indeed, if clients are not identified nor traced and all downloads are made through a PIR protocol, sets of items cannot be singled out. As a consequence, the security requirement for multiple requests holds. Although privacy may not be suitable in all contexts (*e.g.* it can be in contradiction with the interests of the client or of the provider if each accessed content has to be paid for), we believe that a wide range of applications may still benefit from this family of protections.

Yet, we draw attention on the fact that identification and traceability can be achieved through multiple ways (*e.g.* IP address, browser fingerprinting [16], or even side-channel attacks based on dates of download can be enough to trace a behaviour). If even one of them is possible and clients can be singled out, this protection is not sufficient to ensure our multi-query security requirement.

In the case of identification through client accounts, interesting examples include the following:

— *Trusted Third Party:* Relying on a trusted third party to aggregate requests and forwarding it back to clients is a possible solution to ensure privacy. However, it also relies on trusting an external party, which may not always be an easy assumption to reach.

— *Cryptographic Techniques:* Cryptographic techniques, such as group signatures [17], can be used to ensure the validity of the request without revealing the identity of the client. When combined with tools like TOR [18], these tools may provide a very interesting defense to protect the identity of the client.

Perturbation of the number of downloads

Adding noise to the number of downloads, alone or in addition to other techniques, may seem an intuitive solution to our issue, in order to make it hard or impossible to deduce the real value. For instance, by downloading items in which she is not interested in, the client may be able to make some attacks far less efficient. This is all the more

16. Pierre Laperdrix, Nataliia Bielova, et al., « Browser fingerprinting: a survey », *in*: *ACM Transactions on the Web (TWEB)* 14.*2* (2020), pp. 1–33.

17. David Chaum and Eugène Van Heyst, « Group signatures », *in*: *Proc. of EUROCRYPT'91*, Springer, 1991, pp. 257–265.

18. The Onion Rooter is a software providing anonymity of once IP address by forwarding queries through multiple servers randomly `https://www.torproject.org/`

interesting that this decision does not depend on anyone but the client, meaning she can protect herself even without external support. This idea is to be distinguished from the previous one as it does not require to prevent any traceability of the downloads.

However, this protection has very strong limits. First, it increases the load on bandwidth (as more requests than necessary are made), and therefore deteriorates efficiency. Then, adversaries may be able to adapt their model to fit this behaviour (*e.g.* by creating a derived hypergraph knowledge graph that takes the probability of "lying" into account). Last but not least, it is very hard to quantify precisely the quality of protection that is reached. Indeed, usual definitions of anonymization do not apply in this context: k-anonymity [19] seems hard to apply, and most mechanisms for differential privacy [20] can lead to negative noise, while the number of downloads can only be an over-approximation of the number of desired items (it cannot be an under-approximation, as this would mean that the client does not download all desired items). The resulting guarantees would be close to those developed in Aegis [21], but this model assumes both that the client (or her proxy) has access to the hypergraph of knowledge, and that the attacker will either not know the strategy of the client, or not adapt to it.

4.5 Future work

As stated in the introduction, we do not intend here to present a fully finished work on attacks against PIR using external knowledge and the number of downloads, but to expose this issue that has not been, to the best of our knowledge, dealt with in existing works.

We presented in Section 4.4 possible tracks that we envision as counter-measures against these attacks, but a lot of work still has to be made. In particular, we performed no exhaustive experiment nor detailed theoretical analysis on these counter-measures.

Other future works may also include studying the impact of these attacks on other

19. Sweeney, *op. cit.*
20. Dwork, *op. cit.*
21. Victor Zakhary, Ishani Gupta, et al., « Multifaceted Privacy: How to Express Your Online Persona without Revealing Your Sensitive Attributes », *in*: *arXiv preprint arXiv:1905.09945* (2019).

protocols following similar objectives as PIR, such as ORAM [22] [23] [24] [25]. The impact of other side-channel attacks based on other external information is also to be considered. For instance, does using the date of download reveal information about the download itself? If so, does PIR pretend to protect against it or not, and what kind of counter-measures could be used? Exploring blind decoding [26], which allows a client to get the cleartext from a message encrypted with the server's key, without learning the server's secret key nor revealing the cleartext to the server may also be interesting to design new defense solutions.

Another track to explore is to improve our knowledge model: the current knowledge model is interesting, as it allows us to encompass all our use-cases, but it is very heavy (possibly exponential in the number of items), and may be hard to obtain. Lighter models, or efficient methods and heuristics to obtain these models may help designing better and more suitable counter-measures against side-channel attacks.

4.6 Conclusion

In this chapter, we exposed some issues that appear when PIR is used to retrieve multiple items. We presented an attack model, along with an algorithm to directly access some private information that is protected when PIR is used naively to download multiple items. To prevent this leakage, some counter-measures are envisioned, which may adapt to various settings and hypothesis. Although none of these solutions are perfect nor fully tested, we hope that they provide interesting research tracks in the future, to extend a safe use of PIR protocols.

22. Oded Goldreich and Rafail Ostrovsky, « Software protection and simulation on oblivious RAMs », *in*: *Journal of the ACM (JACM)* 43.*3* (1996), pp. 431–473.
23. Martin Maas, Eric Love, et al., « Phantom: Practical oblivious computation in a secure processor », *in*: *Proc. of SIGSAC CCS'13*, 2013, pp. 311–324.
24. Travis Mayberry, Erik-Oliver Blass, and Agnes Hui Chan, « Efficient Private File Retrieval by Combining ORAM and PIR. », *in*: *Proc. of NDSS'14*, 2014.
25. Emil Stefanov, Elaine Shi, and Dawn Song, « Towards practical oblivious RAM », *in*: *arXiv preprint arXiv:1106.3652* (2011).
26. Kouichi Sakurai and Yoshinori Yamane, « Blind decoding, blind undeniable signatures, and their applications to privacy protection », *in*: *Proc. of IH'96*, Springer, 1996, pp. 257–264.

Chapter 5

Conclusion and future work

5.1 Summary of contributions

In this manuscript, we have proposed three main contributions. First, we focused on the protection of privacy in a single-platform setting, using a wide diversity of techniques. Anonymization techniques thanks to random noise, combined with additively homomorphic encryption were used to provide an overview of the distribution of skills among workers. This privacy-preserving distribution can be used to help requesters design tasks more suited to the crowd, or to allow the platform advertising its crowd without endangering the individuals. But we also use this distribution in a second more concrete step: the assignment of tasks to workers. In order to preserve the privacy level expected, we used Private Information Retrieval techniques (with some enhancements) to keep the number of downloads of any given task unknown. Both of these algorithms have been proven secure and experimentally validated: the quality of the distribution and of the assignment, although not perfect due to anonymization and approximations, are reasonable, and the required computation time is also realistically achievable in real-life scenarios.

In the second chapter, the focus was made on regulations in a multi-platform context. In this setting, we proposed two main tools. First a restriction tool, parametrized by law-makers, which regulates the number of interactions allowed between participants (*e.g.* to prevent workers from working more than 40 hours a week, all platforms included, or to prevent monopolies by setting a maximum number of tasks on a given platform). Then, a certification tool, used by participants, allowing them to prove their involvement in tasks, which can be useful for legal actions, social grants, or various legal statuses. Both of these tools are used in a way that preserves privacy as much as possible, while providing transparency by storing traces of all transactions in a public ledger (a permissioned blockchain). Again, these approaches have been both proved and experimentally validated.

Finally, we proposed in the last chapter a work in progress, which aims at drawing attention on an issue with PIR to which we were confronted in the first chapter. To the

best of our knowledge, no related work has documented this issue nor provided solutions to avoid it. Indeed, PIR guarantees that, for a given library of items, the exact item downloaded by a user remains unknown by the uploader. However, this guarantee is only valid for a unique download, and multiple uses of PIR may break the security of the approach. In this last chapter, we proposed a possible attack that can be achieved when multiple uses of PIR are made, by an attacker having access to some external knowledge on the database. Tracks for solutions are also discussed, such as packing queries to avoid leaking their number, distributing the knowledge among participant, providing anonymity to clients, or adding noise to the number of queries. Although none of these approaches have been proved or experimentally validated, we believe that they still provide an interesting first attempt to advertise and maybe even solve this issue.

5.2 Future work

Following these leads, many continuations of our work are possible, either by improving our solutions, or by exploring other crowdsourcing issues. To go on with our solutions, it is still possible, if not necessary, to work on privacy-preserving crowdsourcing platforms: our solution provides an interesting track for general crowdsourcing (*e.g.* not focused on geolocated tasks), but both the quality of our assignment and our computation costs can certainly be improved, while preserving high-level guarantees. Such improvements may come from other partitioning methods than the KD-tree, from randomly sampling the surveyed workers, etc. Other attack models can also be envisioned: honest-but-curious attackers provide an interesting tool, but it is not guaranteed that the platform will comply with the rules all the way (which is the reason for this work). Dealing with covert adversaries [1], that may act maliciously as long as they are not detected, may be a first step toward fully malicious adversaries. In a regulation context, our proposition is mostly a first attempt to explore the combination of privacy, transparency and expressiveness of rules to help law-makers. However, privacy can still possibly be improved and the expressiveness of our model, while sufficient to deal with many real-life scenarios, is still very limited for computer science standards: one may want to enforce conjunctions, disjunctions or negation between constraints, *e.g.* "workers can spend 40 tokens in Uber or in Lyft, but not both", which is currently impossible in our model. Continuations of our work on PIR are pretty straightforward: our intuitions and proposals have to be tested, or submitted to

1. Aumann and Lindell, *op. cit.*

in-depth analysis. As such, we hope that security experts will seize this topic, and help in providing a secure way to download multiple contents while keeping them unidentified from the provider.

Other possible future works may also focus on untreated issues in crowdsourcing. Transparency, accountability and explainability are explicitly described, in the blog of SIGMOD [2], as one of the intellectual challenges of the "Future of Work". In particular, enforcing transparency for all decisions of platforms remains an important matter for workers, as stated by De Stefano [3]. Although mentioned by the same author, portability of profiles or ratings for crowdsourcing has, to the best of our knowledge, been subject to little investigation in current Computer Science literature, and comes at the expense of privacy [4]. Finally, although explored a bit by Elbassuoni et al. [5], fairness in assignment decisions still remains a widely untouched topic, possibly due to the difficulty of designing an absolute definition of fairness in the first place [6].

2. `https://wp.sigmod.org/?p=2931`

3. De Stefano, *op. cit.*

4. Siyuan Han, Zihuan Xu, et al., « Fluid: A blockchain based framework for crowdsourcing », *in*: *Proc. of SIGMOD'19*, 2019, pp. 1921–1924.

5. Shady Elbassuoni, Sihem Amer-Yahia, et al., « Exploring fairness of ranking in online job marketplaces », *in*: 2019.

6. Sorelle A Friedler, Carlos Scheidegger, and Suresh Venkatasubramanian, « On the (im) possibility of fairness », *in*: *arXiv preprint arXiv:1609.07236* (2016).

BIBLIOGRAPHY

Abernathy, James R, Bernard G Greenberg, and Daniel G Horvitz, « Estimates of induced abortion in urban North Carolina », *in*: *Demography* 7.*1* (1970), pp. 19–29.

Agrawal, Rakesh et al., « Order-Preserving Encryption for Numeric Data », *in*: *Proc. of SIGMOD'04*, 2004, pp. 563–574.

Aguilar-Melchor, Carlos et al., « XPIR: Private information retrieval for everyone », *in*: *Proc. of PET'16* 2016.*2* (2016), pp. 155–174.

Allahbakhsh, Mohammad et al., « Quality control in crowdsourcing systems: Issues and directions », *in*: *IEEE Internet Computing* 17.*2* (2013), pp. 76–81.

Allard, Tristan et al., « From Self-Data to Self-Preferences: Towards Preference Elicitation in Personal Information Management Systems », *in*: *Proc. of PAP'17*, (6 pages), 2017.

Amer-Yahia, Sihem and Senjuti Roy, « Toward worker-centric crowdsourcing », *in*: (2016).

Amiri, Mohammad Javad, Divyakant Agrawal, and Amr El Abbadi, « CAPER: a cross-application permissioned blockchain », *in*: *Proc. of the VLDB Endow.* 12.*11* (2019), pp. 1385–1398.

— « SharPer: Sharding Permissioned Blockchains Over Network Clusters », *in*: *arXiv preprint arXiv:1910.00765* (2019).

Amiri, Mohammad Javad et al., *SEPAR: A Privacy-Preserving Blockchain-based System for Regulating Multi-Platform Crowdworking Environments*, Soon submitted to the SIGMOD conference (14 pages), 2020, URL: `https://arxiv.org/abs/2005.01038`.

Andrés, Miguel E et al., « Geo-indistinguishability: Differential privacy for location-based systems », *in*: *Proc. of SIGSAC CCS'13*, 2013, pp. 901–914.

Androulaki, Elli et al., « Hyperledger fabric: a distributed operating system for permissioned blockchains », *in*: *Proc. of EuroSys'18*, 2018, pp. 1–15.

Anwar, Mohammad Amir and Mark Graham, « Between a rock and a hard place: Freedom, flexibility, precarity and vulnerability in the gig economy in Africa », *in*: *Competition & Change* (2019), p. 1024529420914473.

Aumann, Yonatan and Yehuda Lindell, « Security against covert adversaries: Efficient protocols for realistic adversaries », *in*: *Proc. of TCC'07*, Springer, 2007, pp. 137–156.

Bentley, Jon Louis, « Multidimensional binary search trees used for associative searching », *in*: *Communications of the ACM* 18.*9* (1975), pp. 509–517.

Berg, Janine et al., *Digital labour platforms and the future of work : Towards decent work in the online world*, tech. rep., International Labour Organization, 2018.

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

Boldyreva, Alexandra, Nathan Chenette, and Adam O'Neill, « Order-Preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions », *in*: *Proc. of CRYPTO'31*, 2011, pp. 578–595.

Burger-Helmchen, Thierry and Julien Pénin, « Crowdsourcing: définition, enjeux, typologie », *in*: *Management & Avenir 1* (2011), pp. 254–269.

Cachin, Christian, Rachid Guerraoui, and Luís Rodrigues, *Introduction to reliable and secure distributed programming*, Springer Science & Business Media, 2011.

Camenisch, Jan and Jens Groth, « Group signatures: Better efficiency and new theoretical aspects », *in*: *Proc. of SCN'04*, Springer, 2004, pp. 120–133.

Casilli, Antonio A, *En attendant les robots-Enquête sur le travail du clic*, Le Seuil, 2019.

Casilli, Antonio et al., « Le Micro-Travail en France. Derrière l'automatisation, de nouvelles précarités au travail? », *in*: (2019).

Castro, Miguel and Barbara Liskov, « Practical Byzantine fault tolerance and proactive recovery », *in*: *Proc. of TOCS'02* 20.*4* (2002), pp. 398–461.

Cecchetti, Ethan et al., « Solidus: Confidential distributed ledger transactions via PVORM », *in*: *Proc. of SIGSAC CCS'17*, 2017, pp. 701–717.

Celis, L Elisa et al., « Assignment Techniques for Crowdsourcing Sensitive Tasks », *in*: *Proc. of CSCW'16*, 2016, pp. 836–847.

Chase, JP Morgan, *Quorum white paper*, 2016.

Chaum, David and Eugène Van Heyst, « Group signatures », *in*: *Proc. of EUROCRYPT'91*, Springer, 1991, pp. 257–265.

Chen, Bee-Chung, Kristen LeFevre, and Raghu Ramakrishnan, « Privacy skyline: privacy with multidimensional adversarial knowledge », *in*: *Proc. of VLDB'07*, VLDB Endowment, 2007, pp. 770–781.

Chittilappilly, Anand Inasu, Lei Chen, and Sihem Amer-Yahia, « A Survey of General-Purpose Crowdsourcing Techniques », *in*: *IEEE Transactions on Knowledge and Data Engineering* 28.*9* (2016), pp. 2246–2266.

Chor, Benny et al., « Private information retrieval », *in*: *Proc. of FOCS'95*, 1995, pp. 41–50.

Cohen, Aloni and Kobbi Nissim, « Linear Program Reconstruction in Practice », *in*: *CoRR* (2018), arXiv: 1810.05692.

Cohen, Julie E, « Law for the platform economy », *in*: *UC Davis Law Review, Forthcoming* 51 (2017), p. 133.

Cormode, Graham et al., « Differentially private spatial decompositions », *in*: *Proc. of ICDE'12*, 2012, pp. 20–31.

Damgård, Ivan and Mads Jurik, « A generalisation, a simplification and some applications of paillier's probabilistic public-key system », *in*: *Proc. of PKC'01*, 2001, pp. 119–136.

De Stefano, Valerio, « The rise of the just-in-time workforce: On-demand work, crowdwork, and labor protection in the gig-economy », *in*: *Comp. Lab. L. & Pol'y J.* 37 (2015), p. 471.

Devet, Casey and Ian Goldberg, « The best of both worlds: Combining information-theoretic and computational PIR for communication efficiency », *in*: *Proc. of PETS'14*, 2014, pp. 63–82.

Dinur, Irit and Kobbi Nissim, « Revealing information while preserving privacy », *in*: *Proc. of SIGACT-SIGMOD-SIGART'03*, 2003, pp. 202–210.

Doan, Anhai, Raghu Ramakrishnan, and Alon Y Halevy, « Crowdsourcing systems on the world-wide web », *in*: *Communications of the ACM* 54.*4* (2011), pp. 86–96.

Duguépéroux, Joris, « Guaranteed Confidentiality and Efficiency in Crowdsourcing Platforms », *in*: *BDA'17*, (2 pages), 2017.

— « Guaranteed Confidentiality and Efficiency in Crowdsourcing Platforms », *in*: *APVP'19*, (2 pages), 2019.

Duguépéroux, Joris and Tristan Allard, « From Task Tuning to Task Assignment in Privacy-Preserving Crowdsourcing Platforms », *in*: *Transactions on Large-Scale Data and Knowledge-Centered Systems* (2020), (40 pages).

— « Privacy-Preserving Informed Task Design in Crowdsourcing Processes », *in*: *BDA'19*, (10 pages), 2019.

— « Un algorithme de partitionnement de l'espace des compétences pour plateformes de crowdsourcing respectueuses de la vie privée », *in*: *HIA'20*, (17 pages), 2020.

Duguépéroux, Joris, Antonin Voyez, and Tristan Allard, « Task-Tuning in Privacy-Preserving Crowdsourcing Platforms », *in*: *Proc. of EDBT'20*, (4 pages), 2020, pp. 623–626.

— « Task-Tuning in Privacy-Preserving Crowdsourcing Platforms », *in*: *HIA'20*, (8 pages), 2020.

Dwork, Cynthia, « Differential privacy », *in*: *Proc. of ICALP'06*, 2006, pp. 1–12.

Dwork, Cynthia and Aaron Roth, « The algorithmic foundations of differential privacy », *in*: *Foundations and Trends in Theoretical Computer Science* 9.*3–4* (2014), pp. 211–407.

Elbassuoni, Shady et al., « Exploring fairness of ranking in online job marketplaces », *in*: 2019.

Ethereum blockchain app platform, https://www.ethereum.org, 2017.

Finnerty, Ailbhe et al., « Keep it Simple: Reward and Task Design in Crowdsourcing », *in*: *Proc. of SIGCHI'13*, 2013, 14:1–14:4.

Fischer, Michael J, Nancy A Lynch, and Michael S Paterson, « Impossibility of distributed consensus with one faulty process », *in*: *Journal of the ACM (JACM)* 32.*2* (1985), pp. 374–382.

Friedler, Sorelle A, Carlos Scheidegger, and Suresh Venkatasubramanian, « On the (im) possibility of fairness », *in*: *arXiv preprint arXiv:1609.07236* (2016).

Future of Work, Global Commission on the, *Work for a brighter future*, tech. rep., International Labour Organization, 2019.

Geiger, David et al., « Crowdsourcing Information Systems - Definition, Typology, and Design », *in*: *Proc. of ICIS'12*, 2012.

Gentry, Craig, « Fully homomorphic encryption using ideal lattices », *in*: *Proc. of STOC'09*, 2009, pp. 169–178.

Gentry, Craig and Dan Boneh, *A fully homomorphic encryption scheme*, vol. 20, 9, Stanford university Stanford, 2009.

Ghosh, Arpita, Tim Roughgarden, and Mukund Sundararajan, « Universally utility-maximizing privacy mechanisms », *in*: *SIAM Journal on Computing* 41.*6* (2012), pp. 1673–1693.

Goldreich, Oded, « Foundations of Cryptography–A Primer », *in*: *Foundations and Trends® in Theoretical Computer Science* 1.*1* (2005), pp. 1–116.

Goldreich, Oded and Rafail Ostrovsky, « Software protection and simulation on oblivious RAMs », *in*: *Journal of the ACM (JACM)* 43.*3* (1996), pp. 431–473.

Goldwasser, Shafi and Silvio Micali, « Probabilistic encryption », *in*: *Journal of computer and system sciences* 28.*2* (1984), pp. 270–299.

Gray, Mary L and Siddharth Suri, *Ghost Work: How to Stop Silicon Valley from Building a New Global Underclass*, Eamon Dolan Books, 2019.

Gupta, Trinabh et al., « Scalable and Private Media Consumption with Popcorn. », *in*: *Proc. of NSDI'16*, 2016, pp. 91–107.

Han, Siyuan et al., « Fluid: A blockchain based framework for crowdsourcing », *in*: *Proc. of SIGMOD'19*, 2019, pp. 1921–1924.

Hay, Michael et al., « Boosting the accuracy of differentially private histograms through consistency », *in*: *Proc. of the VLDB Endow.* 3.*1-2* (2010), pp. 1021–1032.

Hay, Michael et al., « Principled evaluation of differentially private algorithms using dpbench », *in*: *Proc. of SIGMOD'16*, ACM, 2016, pp. 139–154.

Hopwood, Daira et al., « Zcash protocol specification », *in*: *GitHub: San Francisco, CA, USA* (2016).

Howe, Jeff, « The rise of crowdsourcing », *in*: *Wired magazine* 14.*6* (2006), pp. 1–4.

International Labour Legislation, Commission on, *Constitution of the International Labour Organization*, 1919.

Kajino, Hiroshi, « Privacy-Preserving Crowdsourcing », PhD thesis, Univ. of Tokyo, 2015.

Kajino, Hiroshi, Yukino Baba, and Hisashi Kashima, « Instance-Privacy Preserving Crowdsourcing », *in*: *Proc. of HCOMP'14*, 2014.

Karmarkar, Narendra and Richard M Karp, *The Difierencing Method of Set Partitioning*, tech. rep., Technical Report UCB/CSD 82/113, Computer Science Division, University of California, Berkeley, 1982.

Kellaris, Georgios, Stavros Papadopoulos, and Dimitris Papadias, « Engineering Methods for Differentially Private Histograms: Efficiency Beyond Utility », *in*: *IEEE TKDE* 31.*2* (2018), pp. 315–328.

Kenney, Martin and John Zysman, « The rise of the platform economy », *in*: *Issues in science and technology* 32.*3* (2016), p. 61.

Kifer, Daniel, « Attacks on privacy and deFinetti's theorem », *in*: *Proc. of SIGMOD'09*, 2009, pp. 127–138.

Kosba, Ahmed et al., « Hawk: The blockchain model of cryptography and privacy-preserving smart contracts », *in*: *Proc. of SP'16*, IEEE, 2016, pp. 839–858.

Kucherbaev, Pavel et al., « Crowdsourcing processes: A survey of approaches and opportunities », *in*: *IEEE Internet Computing* 20.*2* (2015), pp. 50–56.

Kulkarni, Anand Pramod, Matthew Can, and Bjoern Hartmann, « Turkomatic: automatic, recursive task and workflow design for mechanical turk », *in*: *Proc. HCOMP'11*, 2011.

Kulkarni, Anand, Matthew Can, and Björn Hartmann, « Collaboratively crowdsourcing workflows with turkomatic », *in*: *Proc. of CSCW'12*, 2012, pp. 1003–1012.

Kushilevitz, Eyal and Rafail Ostrovsky, « Replication is not needed: Single database, computationally-private information retrieval », *in*: *Proc. of FOCS'97*, IEEE, 1997, pp. 364–373.

Lamport, Leslie et al., « Paxos made simple », *in*: *ACM Sigact News* 32.*4* (2001), pp. 18–25.

Laperdrix, Pierre et al., « Browser fingerprinting: a survey », *in*: *ACM Transactions on the Web (TWEB)* 14.*2* (2020), pp. 1–33.

Lease, Matthew et al., « Mechanical Turk is Not Anonymous », *in*: *SSRN Electronic Journal* (2013).

Li, Guoliang et al., « Crowdsourced data management: A survey », *in*: *IEEE TKDE* 28.*9* (2016), pp. 2296–2319.

Li, Ninghui, Tiancheng Li, and Suresh Venkatasubramanian, « t-closeness: Privacy beyond k-anonymity and l-diversity », *in*: *Proc. of ICDE'07*, IEEE, 2007, pp. 106–115.

Lu, Yuan, Qiang Tang, and Guiling Wang, « Zebralancer: Private and anonymous crowdsourcing system atop open blockchain », *in*: *Proc. of ICDCS'18*, IEEE, 2018, pp. 853–865.

Maas, Martin et al., « Phantom: Practical oblivious computation in a secure processor », *in*: *Proc. of SIGSAC CCS'13*, 2013, pp. 311–324.

Machanavajjhala, Ashwin, Johannes Gehrke, and Michaela Götz, « Data publishing against realistic adversaries », *in*: *Proc. of VLDB'07* 2.*1* (2009), pp. 790–801.

Martin, David J et al., « Worst-case background knowledge for privacy-preserving data publishing », *in*: *Proc. of ICDE'07*, IEEE, 2007, pp. 126–135.

Mavridis, Panagiotis, David Gross-Amblard, and Zoltán Miklós, « Using Hierarchical Skills for Optimized Task Assignment in Knowledge-Intensive Crowdsourcing », *in*: *Proc. of WWW'16*, 2016, pp. 843–853.

Mayberry, Travis, Erik-Oliver Blass, and Agnes Hui Chan, « Efficient Private File Retrieval by Combining ORAM and PIR. », *in*: *Proc. of NDSS'14*, 2014.

McSherry, Frank and Kunal Talwar, « Mechanism design via differential privacy », *in*: *Proc. of FOCS'07*, IEEE, 2007, pp. 94–103.

Méda, Dominique, *Le Travail. une valeur en voie de disparition?*, Flammarion, 2010.

Mironov, Ilya et al., « Computational Differential Privacy », *in*: *Proc. of CRYPTO'29*, 2009, pp. 126–142.

Paillier, Pascal, « Public-key cryptosystems based on composite degree residuosity classes », *in*: *Proc. of EUROCRYPT'99*, 1999, pp. 223–238.

Parameswaran, Aditya Ganesh et al., « Deco: declarative crowdsourcing », *in*: *Proc. of CIKM'21*, 2012, pp. 1203–1212.

participants, FoW, *Imagine all the People and AI in the Future of Work*, ACM SIGMOD blog post, 2019.

Private Data Collections: A High-Level Overview, https://www.hyperledger.org/blog/2018/10/23/priva data-collections-a-high-level-overview.

Qardaji, Wahbeh, Weining Yang, and Ninghui Li, « Differentially private grids for geospatial data », *in*: *Proc. of ICDE'13*, 2013, pp. 757–768.

— « Understanding Hierarchical Methods for Differentially Private Histograms », *in*: *Proc. VLDB Endow.* 6.*14* (2013), pp. 1954–1965, ISSN: 2150-8097.

Regulating gig, crowd, and platform work, URL: `https://humancomputerinteraction.w vu.edu/crowd-work/literature-review`.

Sakurai, Kouichi and Yoshinori Yamane, « Blind decoding, blind undeniable signatures, and their applications to privacy protection », *in*: *Proc. of IH'96*, Springer, 1996, pp. 257–264.

Sánchez, David Cerezo, « Raziel: Private and verifiable smart contracts on blockchains », *in*: *arXiv preprint arXiv:1807.09484* (2018).

Schmidt, Florian Alexander, *Crowd Design: From Tools for Empowerment to Platform Capitalism*, Birkhäuser, 2017.

Sion, Radu and Bogdan Carbunar, « On the computational practicality of private information retrieval », *in*: *Proc. of NDSS'07*, 2007, pp. 2006–06.

Srba, Ivan and Maria Bielikova, « A comprehensive survey and classification of approaches for community question answering », *in*: *ACM TWEB* 10.*3* (2016), p. 18.

Srnicek, Nick, *Platform capitalism*, John Wiley & Sons, 2017.

Stefanov, Emil, Elaine Shi, and Dawn Song, « Towards practical oblivious RAM », *in*: *arXiv preprint arXiv:1106.3652* (2011).

Stefanov, Emil et al., « Path ORAM: an extremely simple oblivious RAM protocol », *in*: *Proc. of SIGSAC CCS'13*, 2013, pp. 299–310.

Stern, Julien P, « A new and efficient all-or-nothing disclosure of secrets protocol », *in*: *Proc. of ASIACRYPT'98*, Springer, 1998, pp. 357–371.

Steutel, Fred W and Klaas Van Harn, *Infinite divisibility of probability distributions on the real line*, 2003.

Sweeney, Latanya, « k-anonymity: A model for protecting privacy », *in*: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10.*05* (2002), pp. 557–570.

Taft, Rebecca et al., « E-store: Fine-grained elastic partitioning for distributed transaction processing systems », *in*: *Proc. of the VLDB Endow.* 8.*3* (2014), pp. 245–256.

Tao, Qian et al., « Differentially Private Online Task Assignment in Spatial Crowdsourcing: A Tree-based Approach », *in*: *Proc. of ICDE'20*, IEEE, 2020, pp. 517–528.

Thomson, Alexander et al., « Calvin: fast distributed transactions for partitioned database systems », *in*: *Proc. of MOD'12*, ACM, 2012, pp. 1–12.

To, Hien, Gabriel Ghinita, and Cyrus Shahabi, « A framework for protecting worker location privacy in spatial crowdsourcing », *in*: *Proc. of the VLDB Endow.* 7.*10* (2014), pp. 919–930.

To, Hien, Cyrus Shahabi, and Li Xiong, « Privacy-preserving online task assignment in spatial crowdsourcing with untrusted server », *in*: *Proc. of ICDE'18*, 2018, pp. 833–844.

Wong, Raymond Chi-Wing et al., « Minimality attack in privacy preserving data publishing », *in*: *Proc. of VLDB'07*, 2007, pp. 543–554.

Xia, Huichuan et al., « Our Privacy Needs to be Protected at All Costs: Crowd Workers' Privacy Experiences on Amazon Mechanical Turk », *in*: *Proc. of HCI'17* 1 (2017), p. 113.

Xiao, Xiaokui and Yufei Tao, « M-invariance: towards privacy preserving re-publication of dynamic datasets », *in*: *Proc. of SIGMOD'07*, 2007, pp. 689–700.

Zakhary, Victor et al., « Multifaceted Privacy: How to Express Your Online Persona without Revealing Your Sensitive Attributes », *in*: *arXiv preprint arXiv:1905.09945* (2019).

Zhai, Dongjun et al., « Towards secure and truthful task assignment in spatial crowdsourcing », *in*: *World Wide Web* 22.*5* (2019), pp. 2017–2040.

Zhang, Jun, Xiaokui Xiao, and Xing Xie, « PrivTree: A Differentially Private Algorithm for Hierarchical Decompositions », *in*: *Proc. of SIGMOD'16*, 2016, pp. 155–170.

Zhu, Saide et al., « zkCrowd: a hybrid blockchain-based crowdsourcing platform », *in*: *IEEE Transactions on Industrial Informatics* (2019).

UNIVERSITÉ DE RENNES 1

Titre : Protection des travailleurs dans les plateformes de crowdsourcing : une perspective technique

Mot clés : Crowdsourcing, Protection de la Vie Privée, Confidentialité Différentielle, Systèmes Distribués, Régulation, PIR

Résumé : Ce travail porte sur les moyens de protéger les travailleurs dans le cadre du crowdsourcing. Une première contribution s'intéresse à la protection de la vie privée des travailleurs pour une plateforme unique, tout en autorisant différents usages des données (pour affecter des tâches aux travailleurs ou pour avoir des statistiques sur la population par exemple). Une seconde contribution propose la mise à disposition d'outils, pour les législateurs, permettant de réguler de multiples plateformes en combinant à la fois transparence et respect de la vie privée. Ces deux approches font appel à de nombreux outils (d'anonymisation, de chiffrement ou de distribution des calculs notamment), et sont à la fois accompagnées de preuves de sécurité et validées par des expérimentations. Une troisième contribution, moins développée, propose de mettre en lumière un problème de sécurité dans une des techniques utilisées (le PIR) lorsque celle-ci est utilisée à de multiples reprises, problème jusqu'à présent ignoré dans les contributions de l'état de l'art.

Title: Protecting Workers in Crowdsourcing Platforms: a Technical Perspective

Keywords: Crowdsourcing, Privacy, Differential Privacy, Distributed Systems, Regulation, Private Information Retrieval

Abstract: This work focuses on protecting workers in a crowdsourcing context. Indeed, workers are especially vulnerable in online work, and both surveillance from platforms and lack of regulation are frequently denounced for endangering them. Our first contribution focuses on protecting their privacy, while allowing usages of their anonymized data for, *e.g.* assignment to tasks or providing help for task-design to requesters. Our second contribution considers a multi-platform context, and proposes a set of tools for law-makers to regulate platforms, allowing them to enforce limits on interactions in various ways (to limit the work time for instance), while also guaranteeing transparency and privacy. Both of these approaches make use of many technical tools such as cryptography, distribution, or anonymization tools, and include security proofs and experimental validations. A last, smaller contribution, draws attention on a limit and possible security issue for one of these technical tools, the PIR, when it is used multiple times, which has been ignored in current state-of-the-art contributions.