SEPAR: Towards Regulating Future of Work Multi-Platform Crowdworking Environments with Privacy Guarantees

Anonymous Author(s)

Abstract

Crowdworking platforms provide the opportunity for diverse workers to execute tasks for different requesters. The popularity of the "gig" economy has given rise to independent platforms that provide competing and complementary services. Workers as well as requesters with specific tasks may need to work for or avail from the services of multiple platforms resulting in the rise of *multi-platform crowdworking systems*. Recently, there has been increasing interest by governmental, legal and social institutions to enforce regulations, such as minimal and maximal work hours, on crowdworking platforms. Platforms within multi-platform crowdworking systems, therefore, need to collaborate to enforce cross-platform regulations. While collaborating to enforce global regulations requires the *transparent* sharing of information about tasks and their participants, the *privacy* of all participants needs to be preserved. In this paper, we propose an overall vision exploring the regulation, privacy, and architecture dimensions for future of work multi-platform crowdworking environments. We then present *SEPAR*, a multi-platform crowdworking system that enforces a large sub-space of practical global regulations on a set of distributed independent platforms in a privacy-preserving manner. SEPAR, enforces *privacy* using *lightweight and anonymous tokens*, while *transparency* is achieved using fault-tolerant *blockchains* shared across multiple platforms. The privacy guarantees of SEPAR against covert adversaries are formalized and thoroughly demonstrated, while the experiments reveal the efficiency of SEPAR in terms of performance and scalability.

1 Introduction

The rise of the "gig" or *platform economy* [14, 18] is reshaping work all around the world. Crowdsourcing platforms dedicated to work (also called *crowdworking platforms* [7]) are online intermediaries between *requesters* and *workers*, where requesters propose *tasks* while *workers* propose skills and time. By providing requesters (resp. workers) 24/7 access to a worldwide workforce (resp. worldwide task market), crowdworking platforms have grown in numbers, diversity, and adoption. Today, crowdworkers come from countries spread all over the world, and work on several, possibly competing, platforms [7]. The use of crowdworking platforms is expected to continue growing [27], and in fact they are envisioned as key technological components of the future of work [8, 15, 28].

Crowdworking platforms, however, challenge national boundaries and weaken the formal relationships between the platforms, workers and task requesters. Guaranteeing the compliance of crowdworking platforms with national or regional labour laws is hard[1] [27] despite the stringent need for regulating work. For example, the total work hours of a worker per week may not exceed 40 hours to follow *Fair Labor Standards Act*[2] (FLSA). In California, Assembly Bill 5 (*AB5*)[3] entitles workers to greater labor protections, such as minimum wage laws, sick leave, and unemployment and workers' compensation benefits. *AB5* is currently being challenged by *California Proposition 22*[4], which also imposes its own set of regulations on minimal hours worked for health benefits. 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 crowdworking platforms. While legal tools are currently being investigated [26, 27], there is a stringent need for technical tools allowing official institutions to enforce regulations.

Some platforms have already started implementing self-defined *local regulations*. For example, Uber[5] and Lyft[6] force drivers to rest at least 6 hours for every 12 hours in driver mode, or Wirk.io[7] prevent micro-workers from earning more than €3000 per year. However, since workers and requesters can simply switch platforms when a local limit is reached, no global cross-platform regulation can be enforced. Moreover, participants in a crowdworking task may also behave maliciously or act as adversaries, e.g., violate the privacy of participants or the regulations for their benefits. The privacy of participants and the global consistency of regulations are considered as critical needs for future of work crowdworking environments [8].

Most current crowdworking platforms are independent of each other. However, the emergence of complex tasks that may need multiple contributions from possibly different platforms, on one hand, and more importantly, the enforcement of legal regulations, on the other hand, highlights the need for collaboration between crowdworking platforms, thus resulting in *multi-platform crowdworking systems*. For example, many drivers work for both Uber and Lyft concurrently[8], while a requester may also request multiple rides from both Uber and Lyft concurrently. The observation holds also for microtask platforms [7] as well where a requester who has registered on *Amazon Mechanical Turk* and *Prolific* might need hundreds of contributions for a single microtask at the same time and 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 while ensure minimal and maximum hourly regulations on drivers without revealing any private information about the workers to competing platforms.

Our overall vision for multi-platform crowdworking environments needs to address three main dimensions: *regulations*, *privacy*,

[1]See, e.g., the `Otey V Crowdflower` class action against a famous microtask platform for *"substandard wages and oppressive working hours"* (casetext.com/case/otey-v-crowdflower-1).

[2]https://www.dol.gov/agencies/whd/flsa

[3]https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201920200AB5

[4]https://ballotpedia.org/California_Proposition_22,_App-Based_Drivers_as_Contractors_and_Labor_Policies_Initiative_(2020)

[5]https://www.uber.com/en-ZA/blog/driving-hours-limit/

[6]https://help.lyft.com/hc/articles/115012926787-Taking-breaks-and-time-limits

[7]https://www.wirk.io/50k-freelances-en-france/

[8]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/.

and *architecture*. First, a multi-platform crowdworking system must clearly define the *types of regulations* supported in terms of the *complexity* of the regulation (e.g., a simple or a chain of interactions) as well as the *aggregate* requirements of the regulation (e.g., no aggregation or SUM aggregations). For example, California Proposition 22, which states "if a driver works at least 25 hours per week, companies (i.e., platforms) require to provide healthcare ...", is a simple, SUM-aggregate regulation. Second, the threat model (e.g., *honest-but-curious*, *covert*, *fully malicious*) as well as the privacy guarantees provided to each entity must be clearly specified. Finally, from an architecture design point of view, any multi-platform crowdworking environment consists of two critical components: *regulation management* that models and enforces regulations and *global state management* that manages the global states of all participants as well as tasks. Each of these components can be implemented in a centralized or a decentralized approach.

In this paper, we present *SEPAR*, a possible instance of a precise point in the design space of multi-platform crowdworking systems. SEPAR results from choices, guided by cutting edge real-life regulation proposals, on all three regulations, privacy and architecture dimensions of the design space. First, SEPAR focuses on managing lower and upper bounds on aggregate-based regulations. Second, SEPAR considers that any participant in a crowdworking environment may act as a *covert adversary* [6] and ensures that no participant obtains or infers any information about a crowdworking task beyond what is strictly needed for accomplishing its local crowdworking task and for the distributed enforcement of regulations. Finally, in SEPAR, we opted for simplicity and rapid prototyping to use a centralized (but fault-tolerant) component to manage regulations, however, a decentralized component to manage the global state of the system. In particular, SEPAR uses a *permissioned blockchain* as an underlying infrastructure that is shared among all involved platforms.

The complexity of the conjunction of the required properties makes the problem non-trivial. First, the regulations need to be expressed in a *simple* and *non-ambiguous* manner. Second, while enforcing regulations over multiple crowdworking platforms requires the global state of the system to be *transparent*, the *privacy* of participants needs to be preserved, hence SEPAR needs to reconcile *transparency* with *privacy*. Finally, the decentralized management of the global state among a distributed set of crowdworking platforms requires distributed consensus protocols.

SEPAR is a two-level solution consisting of a *privacy-preserving token-based system* (i.e., the application level) on top of a *blockchain ledger* shared across platforms (i.e., the infrastructure level). First, at the application level, global regulations are modeled using *lightweight and anonymous tokens* distributed to workers, platforms, and requesters. The information shared among platforms and participants is limited to the minimum necessary for performing the tasks against *adversarial participants acting as covert adversaries*. Second, at the infrastructure level, the blockchain ledger allows SEPAR to provide transparency across platforms. Nonetheless, for the sake of privacy and to improve performance, the ledger is *not maintained* by any platform and each platform maintains only a view of the ledger. We then design a suite of consensus protocols for coping with the concurrency issues inherent to a multi-platform context. Salient features of SEPAR include the simplicity of its building blocks (e.g., usual signature schemes) and its compatibility with today's platforms (e.g., it does not jeopardize their privacy requirements of requesters and workers for enforcing the regulation).

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

(1) A vision for the design space of regulation systems for future of work multi-platform crowdworking environments. In particular, (1) we express regulations as SQL constraints and categorize them according to their SQL expression, (2) we propose a formal privacy model for multi-platform regulated crowdworking system based on the well-known simulatability paradigm, and (3) we discuss critical components of their architecture.
(2) SEPAR, a two-level *privacy-preserving transparent* multi-platform proof-of concept crowdworking system that enforces a precise point in the design space guided by cutting edge practical regulations currently discussed by societal organizations, legal entities and enterprises. SEPAR ensures privacy using *lightweight and anonymous tokens*, while transparency is achieved using a *blockchain* shared across platforms for both crash-only and Byzantine nodes.
(3) A formal security analysis of SEPAR and thorough experimental evaluations.

The paper is organized as follows. The overall vision of the design space for regulation systems is presented in Section 2. Section 3 presents the SEPAR model. The implementation of regulations in SEPAR is discussed in Section 4. The SEPAR blockchain ledger and consensus protocols are presented in Section 5. Section 6 details a thorough experimental evaluation, Section 7 discusses the related work, and finally, Section 8 concludes the paper.

2 Design Space of Regulation Systems

Designing a system for regulating crowdworking platforms mandates three main choices. First, given the large variety of regulations that apply to working environments, any given regulation system must clearly define the *types of regulations* it supports. Second, a crowdworking environment involves a set of distributed entities (platforms, workers, requesters) that cannot be fully trusted. The *privacy guarantees* provided to each entity must be rigorously stated (e.g., computational). Third, the distributed nature of a crowdworking environment requires various *architectural choices* that range from fully decentralized approaches (e.g., peer-to-peer architectures) to the traditional centralized architecture. In this section, we first define a crowdworking environment and then characterize the design space for crowdworking regulation systems.

2.1 Crowdworking Environment

A *crowdworking environment* consists of a set of workers $\mathcal{W}$ interacting with a set of requesters $\mathcal{R}$ through a set of competing platforms $\mathcal{P}$. We refer to the workers, platforms, and requesters of a crowdworking environment as *participants*. 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 hand, 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 interactions between workers and requesters. A *crowdworking process* π connects three participants – a workers w, a platform p, and a requester r – and aims to facilitate the execution of a task $t \in \mathcal{T}_r$ through platform p via a worker w. For simplicity and without loss of generality, we assume that each process corresponds to a time unit of work (e.g., 1 hour).

2.2 Types of Regulation

The space of possible regulations can be structured based on two orthogonal dimensions: the *complexity* of the regulation, e.g., constraints on a single process versus constraints that apply to multiple processes that are transitively related, and the *aggregate* nature of the regulation, e.g., no aggregation versus restrictions involving SUM aggregate.

In order to give a clear semantics to each dimension and to rigorously map regulations to points in this space, we express regulations by SQL constraints over a *universal virtual table* storing information about all crowdworking processes having been performed. We denote this table by U-TABLE and emphasize that it is *virtual* (we use it only for clarifying the various types of regulations, it is never instantiated). The attributes of U-TABLE refer to meta-data about the interactions (i.e., at least the worker, requester, and platform involved in a process, and also possibly additional metadata such as begin and end timestamps) and information about the contents (e.g., the time estimate for the task[9] - the TIMECOST attribute below -, the proposed wage, the worker's contribution). For simplicity, we focus below on the attributes of the U-TABLE relevant to our illustrative examples: (1) TS_BEGIN, TS_END, WORKER, PLATFORM, and REQUESTER, and (2) TIMECOST, WAGE, and CONTRIBUTION. Finally, a regulation is simply a Boolean SQL expression nested within the usual CHECK clause: ALTER TABLE U ADD CONSTRAINT r CHECK (...).

The complexity and aggregate dimensions of a regulation can both be deduced from its SQL expression. The complexity is given by the presence of *join* operations while the aggregate dimension is given by the presence of *aggregate* function(s), possibly with GROUP BY and HAVING clause(s). For simplicity, we consider a coarse grain characterization of these two dimensions. A regulation is *simple* if there is no join and *complex* otherwise. A regulation is *row-only* if it does not involve any aggregate function, aggregate-only if it involves only comparison(s) over aggregate(s), and *mixed* if it involves comparisons over rows and aggregates.

We illustrate the possible types of regulations based on simple examples extracted from real-life crowdworking regulations or from real-life proposals of regulation. First, we consider a regulation r_1 requiring the wage proposed by each task to be at least a given amount θ. This regulation is similar to CA Proposition 22. It illustrates the (simple, no-aggregate) type of regulation.

```
ALTER TABLE U-TABLE ADD CONSTRAINT r1 CHECK (
  NOT EXISTS (
    SELECT * FROM U
    WHERE TIMECOST ≤ θ
  ) );
```

[9] Future regulation systems will need to design technical means to guarantee the reliability of the estimations of tasks (e.g., privacy-preserving feedback systems from workers, automatic time estimation by analyzing task descriptions).

Second, we consider a regulation r_2 requiring each worker to work at most a given amount of time units θ per time period ρ. It illustrates the (simple, SUM-aggregate) type of regulation. It is similar to the regulation followed by the wirk.io platform that limits the crowdworking gains of any worker on the platform to €3000 per year. The following SQL constraint expresses r_2, assuming that current_time() gives the current time in the same unit as the period ρ.

```
ALTER TABLE U-TABLE ADD CONSTRAINT r2 CHECK (
  NOT EXISTS (
    SELECT * FROM U
    WHERE WORKER=w AND current_time()-TS_BEGIN ≤  ρ
    GROUP BY WORKER
    HAVING SUM(TIMECOST) ≥  θ
  ) );
```

Finally, we complete our illustrations by considering a regulation r_3 that prevents any worker to submit two similar contributions to the same requester (even through two distinct platforms). We assume that the sim function computes the similarity between two contributions and that θ is the threshold above which we consider that two contributions are too similar. This illustrates the (complex, no-aggregate) type of regulation.

```
ALTER TABLE U-TABLE ADD CONSTRAINT r3 CHECK (
  NOT EXISTS (
    SELECT *
    FROM U U1 JOIN U U2 ON
        U1.WORKER=U2.WORKER
        AND U1.REQUESTER=U2.REQUESTER
        AND sim(U1.CONTRIBUTION, U2.CONTRIBUTION) ≥  θ
  ) );
```

Although most regulations must always hold (e.g., a lower than constraint on a (simple, SUM-aggregate) regulation, a (complex, no-aggregate) regulation), some regulations, inherently, *cannot* always hold. Similar to deferred SQL constraints, they must only hold after a given time period. For example, a periodic greater than constraint on a (simple, SUM-aggregate) regulation cannot hold initially, but must hold at the end of a given period (e.g., CA Proposition 22 that requires a worker to work at least 25 hours per week to qualify for healthcare subsidies). We call *enforceable* the regulations that must always hold and *verifiable* the regulations that eventually hold. The verifiable/enforceable property of a regulation is only due to its nature not to its implementation (but it impacts it directly). Future crowdworking regulation systems need to determine the enforceable/verifiable properties of the regulations they support.

2.3 Threat Model and Privacy Model

Crowdworking environments are open environments that connect possibly adversarial participants. Any regulation system must clearly specify both the threat model (e.g., *honest-but-curious*, *covert*, *fully malicious*) and the privacy model. While the threat model only depends on the underlying system, the privacy model can be based on a common formal requirement parameterized for each system by the leaks it tolerates. The possible leakage ranges from no information leaked to any participant (similar to usual *secure multi-party computation* algorithms) to full disclosure (of all the information discussed above) to all participants (e.g., in current crowdworking platforms, the underlying system often requires full disclosure to the platform). The disclosures tolerated by future regulation systems will fall within this range.

We formalize below a common privacy model based on the well-known simulatability paradigm often used by secure multi-party computation algorithms. The proposed model guarantees that nothing leaks, with computational guarantees[10], except the *pluggable* system-dependent tolerated *disclosures*.

Consider a crowdworking process π between worker w, platform p, and requester r for solving a task t. The task may have been sent to several platforms and, depending on the underlying crowdworking platforms, might have been accessed by several workers before being picked and solved. The information generated by the execution of π consists, similarly to the information captured by the U-TABLE, of information about interactions (e.g., at least w, r, and p, the participants directly involved in π) and information about contents (e.g., description of t, contribution, proposed wage). We propose to define the sets of disclosure according to the involvement of a participant in π. Indeed, the platforms not involved in π (i.e., different from p) but that received t may need to learn that t has been completed (e.g., to manage their local copy of the task). Similarly, workers who are not involved in a task t might still need to know that it has been executed, while potentially preserving the privacy of worker w who executed the task. The three sets of disclosures defined below cover all the cases. Future regulation systems have to specify clearly the content of each disclosure set.

- Disclosures to the participants that are **not involved** in π and that have **not received** task t from requester r: $\delta^{\pi}_{\neg R \neg I}$
- Disclosures to the platforms and workers that have **received** the task t from r but that are **not involved** in π: $\delta^{\pi}_{R \neg I}$
- Disclosures to the participants that are directly **involved** in π (and have thus **received** task t): δ^{π}_{RI}

Definition 2.1. Let Π be a set of crowdworking 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 A, the sets of disclosures $(\delta^{\pi}_{\neg R \neg I}, \delta^{\pi}_{R \neg I}, \delta^{\pi}_{RI})$, assuming 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 cp executes the crowdworking process π of ς: $\mathtt{REAL}_{\varsigma,\mathtt{A}(\chi,\delta^{\pi}_{i})}(\mathcal{W},\mathcal{P},\mathcal{R},\mathcal{T}) \overset{c}{\equiv} \mathtt{IDEAL}_{\mathtt{cp},\mathtt{A}(\chi,\delta^{\pi}_{i})}(\mathcal{W},\mathcal{P},\mathcal{R},\mathcal{T})$ where $i \in \{\neg R \neg I, R \neg I, RI\}$, and REAL denotes the adversarial knowledge in the real setting and IDEAL its counterpart in the ideal setting.

2.4 Architecture

The architecture of a multi-platform crowdworking system consists of two main building blocks: *Regulation Management* and *Global State Management*. Regulation management *models* the regulations among the participants and *ensures* that the specified regulations are adhered to by all participants. Global state management, on the other hand, stores the global states of the system including all information relating to the participants and tasks. To implement these two components, similar to all distributed systems, either a centralized or a decentralized approach can be employed. Centralization is typically easier to rapid prototype, while requiring additional technologies to ensure fault-tolerance, privacy, and trustworthiness. A decentralized approach, on the other hand, is more compatible with the setting consisting of diverse organizational entities, while resulting in more overhead and complex communication protocols.

The verifiable/enforceable property of a regulation is another architectural challenge. While enforceable regulations could be enforced within the multi-platform system, verifiable regulations might be of interest to an outside entity, e.g., legal courts or insurance companies. In the latter case, the system needs to provide evidence to an outside entity demonstrating that the regulation was adhered to and hence resolve any disputes that may arise.

3 SEPAR: Design Choices

We propose SEPAR as a possible instance of a precise point in the design space of regulation systems. SEPAR results from choices, guided by cutting edge real-life regulation proposals, on the three dimensions of the design space: supported regulations, disclosures tolerated, and architecture.

3.1 Supported Regulations

SEPAR focuses on enforcing lower and upper bounds on the aggregated working time spent on crowdworking platforms[11], a consensual societal need. The necessary information are the participants to crowdworking processes and the (discrete) time estimation of tasks[12]: for SEPAR, the U-TABLE thus consists of the WORKER, PLATFORM, REQUESTER, and TIMECOST attributes. SEPAR does not consider joins. As a result the kind of regulations supported by SEPAR is (simple, SUM-aggregate). The enforceable/verifiable nature of the regulations supported by SEPAR are easy to determine : $((w,p,r),<,\theta)$ regulations are enforceable because the upper-bound guarantee always hold, and $((w,p,r),>,\theta)$ regulations are verifiable because the lower-bound guarantee cannot always hold (but will have to hold finally, e.g., at the end of each time period).

For simplicity, we propose to express such regulations by (1) a triple (w,p,r) that associates a worker w, a platform p, and a requester r, (2) a comparison operator $<$ or $>$, and (3) a threshold value θ (an integer) that defines the lower/upper bound that needs to hold. Intuitively, a regulation $((w,p,r),<,\theta)$ states that there must not be more than θ time spent by worker w on platform p for requester r. 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[13]. For example $(*,p,r)$ means that the regulation applies to the pair (p,r). A triple may contain up to three $*$ wildcards. An element of a triple that is not a $*$ wildcard is called a *target* of the regulation. Second, the $\forall$ wildcard allows a regulation to express a constraint that must hold for all participants in the same group of participants[14]. For example, $(\forall,p,r)$ represents the following set of triples: $\{(w,p,r)\}, \forall w \in \mathcal{W}$.

[10] The majority of data protection techniques used in real-life are based on encryption schemes that provide guarantees against computationally-bounded adversaries but the model can be easily adapted to information theoretic attackers.

[11] Regulating the wages earned through crowdworking platforms can be dealt with similarly.

[12] Extending regulations with validity periods (e.g., "one week"), is straightforward.

[13] Intuitively, the $*$ wildcard means "whatever".

[14] Intuitively, the $\forall$ wildcard means "for each".

Examples. The semantics of an enforceable regulation without any wildcard, e.g., $e \leftarrow ((w, p, r), <, 26)$ expressing a higher bound on the number of time units spent by worker w for requester r on platform p, is the same as the following SQL query:

```
ALTER TABLE U-TABLE ADD CONSTRAINT e CHECK (
  NOT EXISTS (
    SELECT * FROM U-TABLE
    WHERE WORKER=w AND PLATFORM=p AND REQUESTER=r
    GROUP BY WORKER, PLATFORM, REQUESTER
    HAVING SUM(TIMECOST) ≥ 26
  ) );
```

The weekly FLSA limit on the total work hours per worker can easily be expressed as $r_1 \leftarrow ((\forall, *, *), <, 40)$. A social security institution can request each worker w applying for insurance coverage to prove that she worked more than 5 hours: $r_2 \leftarrow ((w, *, *), >, 5)$ is both necessary and sufficient. Similarly, the regulation $r_3 \leftarrow ((*, p, *), >, 1000)$ allows a tax institution to require from each platform p applying for a tax refund that the total work hours of all its workers is at least 1000 hours.

3.2 Threat Model and Disclosure Sets

SEPAR considers that any participant in a crowdworking environment (worker, requester, platform) may act as a *covert adversary* [6] 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. SEPAR additionally assumes that each crowdworking process involves at least one participant that does not collude with the others[15].

Consider a crowdworking process π between worker w, platform p, and requester r for solving task t. The information generated by the execution of π consists of the relationship between the three participants (w, p, and r) with the task t. Additionally, we also consider the information that π is starting or ending through a starting event BEGIN and an ending event END. They may be determined, for example, by exchanging messages among the participants of π, and may include additional concrete information, e.g., timestamps, IP address. $(\mathsf{BEGIN}, \mathsf{END}, w, p, r, t)$ denotes the information generated by π. SEPAR 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 $\{\mathsf{BEGIN}, \mathsf{END}\}$ events and of the platform p to all participants, whatever their involvement in π. This allows platforms to share information for enforcing regulations (e.g., check that all participants satisfy the regulations before executing π), and to collaborate for correctly managing cross-platforms tasks. Note that for simplicity we use the same notation δ for disclosures concerning *sets* of crowdworking processes as well.

The resulting disclosure sets of SEPAR are instantiated as follows:

- The participants that are **not involved** in π and that have **not received** task t from requester r must not learn anything about the worker, the task, and the requester involved in π: $\delta^{\pi}_{\neg R \neg I} = (\mathsf{BEGIN}, \mathsf{END}, p)$
- The platforms and workers that have **received** task t from r but that are **not involved** in π 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: $\delta^{\pi}_{R \neg I} = (\mathsf{BEGIN}, \mathsf{END}, p, r, t)$
- The participants that are directly **involved** in π (and have thus **received** task t) learn the complete 6-tuple: $\delta^{\pi}_{RI} = (\mathsf{BEGIN}, \mathsf{END}, w, p, r, t)$

[15]A crowdworking process involving only colluding participants can simply take place outside the regulated environment.

3.3 Architecture

SEPAR has two main components. The centralized *Registration Authority (RA)* and the decentralized *Multi-Platform Infrastructure (MPI)*. Although centralized, the RA can be made fault-tolerant using standard replication [20] techniques. RA registers the participants to the crowdworking environment, models the regulations, and distributes to participants the cryptographic material necessary for enforcing or verifying regulations in a secure manner.

MPI, on the other hand, is a decentralized component that maintains the global state of the system including all operations performed by the participants. This state is maintained within a distributed persistent transparent *ledger*. MPI consists of a set of collaborating crowdworking platforms connected by an asynchronous distributed network. Due to the unique features of blockchains such as transparency, provenance, and fault tolerance, the MPI is implemented as a *permissioned blockchain*.

MPI processes two types of tasks: *internal* and *cross-platform*. Internal tasks are submitted to the ledger of a single platform, whereas cross-platform tasks are submitted to the ledgers of multiple platforms (i.e, involved platforms). SEPAR does not make any assumptions on the implementation of crowdworking processes by platforms, e.g., task assignment algorithm and workers contribution delivery. However, processing a task (either internal or cross-platform) requires agreement from the involved platform(s). To establish agreement among the nodes within or across platforms, SEPAR uses *local* and *cross-platform* consensus protocols. Furthermore, in both internal and cross-platform tasks, SEPAR enables all platforms to check the fulfillment of regulations using a *global* consensus protocol among all platforms (irrespective of whether they are involved in the execution of task).

4 SEPAR: Implementation of the Regulations

Inspired by e-cash systems, SEPAR implements both enforceable and verifiable regulations by managing two *budgets* per participant while guaranteeing both privacy and correctness. The overall system is conceptually simple and only relies on the correct use of indidivual and group signatures. The registration authority (RA, see Section 3.3) bootstraps SEPAR, its participants, and refreshes their budgets periodically, but is not involved in the continuous execution of crowdworking processes and their regulations.

4.1 Individual and Group Signatures

Workers, requesters, and platforms are all equipped by the registration authority with the following cryptographic material: a pair of *individual* public/private asymmetric keys (e.g., RSA) and a pair of public/private asymmetric *group* keys (e.g., [9]) where the union of all workers forms a group (in the sense of group signatures), the union of all requesters forms another group, and the union of all platforms forms the last group. A group signature scheme respects three main properties [13]: (1) only members of the group can sign messages, (2) the receiver of the signature can verify that it is a valid signature for the group but cannot discover which member of the group computed it, and (3) in case of dispute later on, the signature can be "opened" (with or without the help of the group members) to reveal the identity of the signer. A common way to enforce the third property is to rely on a *group manager* that can add new members to the group or revoke the anonymity of a signature. Instances of such schemes are proposed in [13], but also in [5, 9]. In SEPAR, we use the protocol proposed in [9] and denote $\sigma^g_w(m)$ the group signature of the worker w (with the private key of her group) for message m. We use equivalent notations for requester r and platform p (i.e., respectively $\sigma^g_r(m)$ and $\sigma^g_p(m)$). The notation $\sigma^i_w(m)$ is used to refer to a individual asymmetric signature (e.g., RSA) of worker w (with her individual (non group) private key) for the message m. We use equivalent notations for requester r and platform p (i.e., respectively $\sigma^i_r(m)$ and $\sigma^i_p(m)$). The registration authority is the group manager for the three groups (workers $\mathcal{W}$, requesters $\mathcal{R}$, platforms $\mathcal{P}$) and is also equipped with her own individual cryptographic keys for signing her messages : $\sigma^i_{RA}(m)$.

4.2 A Simple Token-Based System

To regulate crowdworking processes, our token-based system is defined by five functions: GENERATE for initializing the budgets with the correct number of tokens and refilling them, SPEND for spending portions of the budgets, PROVE for providing proofs for verifying verifiable regulations (e.g., to a third party), CHECK for checking whether a given spending is allowed or not, and ALERT for reporting dubious spending. Since the execution of these functions changes the global state of the system, the data involved in the execution (e.g., tasks, tokens, signatures) must be appended to the distributed ledger of the platforms.

The GENERATE Function. The registration authority uses the GENERATE function to create tokens for all participants (i.e., workers, platforms, requesters) according to the set of regulations. We call e-tokens the tokens implementing enforceable regulations and v-tokens the tokens implementing verifiable regulations[16]. For each enforceable regulation $((w, p, r), <, \theta + 1)$, the registration authority generates θ e-tokens and sends a copy of each token to each target of the regulation. An e-token consists of a *public component* - i.e., 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 by the registration authority[17] - and a *private component* - an index for selecting the correct set of tokens given the other targets, i.e., their public keys[18]. Let e-τ be an e-token, e-τ_{pub} be its public component, e-τ_{priv} be its private component, N be a nonce and λ the list of public keys of the targets of the corresponding regulation. The e-token is thus the pair (e-τ_{pub},e-τ_{priv}) where e-$\tau_{pub} = (N, \sigma^i_{RA}(N))$ and e-$\tau_{priv} = \lambda$. Similar to an e-token, a v-token consists of a public and a private component. The public component is a nonce together with its signature from the registration authority. The private component is simply a triplet of signatures, from the registration authority, binding the recipient of the v-token to each of the other targets of the verifiable regulation. More formally, let v-τ be a v-token, v-τ_{pub} be its public component, v-τ_{priv} be its private component, N be a nonce, o be the identity of the participant owner of the token, and (w, p, r) be the related triplet. The v-token is thus the pair (v-τ_{pub},v-τ_{priv}) where v-$\tau_{pub} = (N, \sigma^i_{RA}(N))$ and v-$\tau_{priv} = (\sigma^i_{RA}(N, o, w), \sigma^i_{RA}(N, o, p), \sigma^i_{RA}(N, o, r))$. The number of v-tokens to produce initially can be easily deduced from the lowest higher bound in enforceable regulations.

The SPEND Function. Requesters create and send their tasks to a platform. The platform appends the tasks to either its own ledger (for internal tasks) or the ledgers of all involved platforms (for cross-platform tasks). Once the task t 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 crowdworking process π starts with the SPEND function, performed as follows. Without loss of generality, we assume below that the given task has a time cost equal to 1. The SPEND function essentially (1) gathers a sufficient number of e-tokens and v-tokens from the participants involved in π, (2) generates the group signatures of the public components of tokens by each participant, and (3) commits the public components of tokens (τ_{pub} for both e-tokens and v-tokens) with the corresponding group signatures $((\sigma^g_w(\tau_{pub}||t), \sigma^g_r(\tau_{pub}||t), \sigma^g_p(\tau_{pub}||t))$ for both e-tokens and v-tokens) to the ledgers of all platforms. Additional signatures are exchanged to guarantee that all the participants involved in π behave honestly (see the extended version for details).

The PROVE function. Participants use the PROVE function to provide proofs for guaranteeing verifiable regulations (e.g., to a third party). The use of v-tokens is relatively straightforward. During the crowdworking processes, participants simply store the private components of v-tokens and deliver them at the end of the validity periods of verifiable regulations. As an example, for a $((w, p, r), >, 4)$ verifiable regulation, the worker w sends the private components of 5 v-tokens. The entity in charge of guaranteeing the verifiable regulation checks the signature of the registration authority - to verify that the participant was involved in the task - and the nonce stored in the ledger - to ensure that the token has been indeed spent and committed to the ledgers of all platforms.

The CHECK and ALERT Functions. These 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 global consensus. Verifications (3) and (4) are similar, and we explain here case (3). When a token is appended to the ledgers of all platforms, any participant (whether involved in the corresponding crowdworking process or not) can CHECK its nonce. If a participant detects a nonce that was received from the registration authority but not spent (or spent on a wrong task[19]), she ALERTs the registration authority. The registration authority will de-anonymize the group signature of the corresponding participant

[16] Although it is technically possible to use a unified token structure for both enforceable and verifiable regulations, using two different tokens reduces computation and communication costs.

[17] Extending tokens with labels and timestamps to support validity periods is straightforward.

[18] 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.

[19] For example, a platform p can collude with a worker w_1 to spend a e-token dedicated to a $(w_2, p, *)$ regulation.

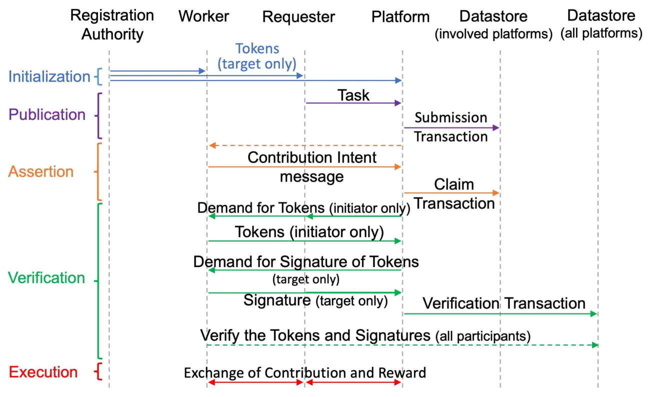

Registration Authority
Worker
Requester
Platform
Datastore (involved platforms)
Datastore (all platforms)
Initialization
Tokens (target only)
Publication
Task
Submission Transaction
Assertion
Contribution Intent message
Claim Transaction
Verification
Demand for Tokens (initiator only)
Tokens (initiator only)
Demand for Signature of Tokens (target only)
Signature (target only)
Verification Transaction
Verify the Tokens and Signatures (all participants)
Execution
Exchange of Contribution and Reward

Figure 1: Sequence chart (references to targets include all participants for v-tokens)

(e.g., the worker's group signature if the alert comes from a worker) and check whether it has been signed by the same participant that sent the token. Depending on the result, the registration authority will take adequate sanctions against the fraudulent participant (true positive) or the alert-riser (false positive). Similarly, if a malicious platform simulates a crash, it can be detected by the lack of updates on the ledger, and proven fraudulent using the logs.

4.3 Task Processing Sequence

The processing of a crowdworking task involves the following five main phases, as depicted in Figure 1.

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 than one platform (i.e., a cross-platform task), the involved platforms collaborate with each other to create a common instance of the task (e.g., a common task identifier). The involved platforms then append the task to their ledgers 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 appends the contribution intent to its ledger 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) and append the claim transaction to their ledgers. If the desired number of contribution for the task has been achieved, the contribution intent is refused.

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 (see above). Upon receiving all tokens and signatures, the platform shares them with all platforms and the e-tokens and their signatures are appended to the ledgers of all platforms through verification transactions. From this point, anyone can check the validity of requirements with the CHECK function (and ALERT if required), as developed above.

Execution. Once all parties have checked the validity of the task, the tokens, and the group signatures, the contribution can be delivered to the requester and the reward to the worker.

4.4 Privacy Analysis

We show below in the suite of Theorem 1, Lemma 1, Lemma 2, and Theorem 2 that the global execution of SEPAR satisfies the δ^{Π}*-privacy* model against covert adversaries (where the disclosure sets are defined in Section 3). All the proofs are included in the extended version.

First, Theorem 1 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 (where the disclosure sets are defined in Section 3).

Theorem 1. (*Privacy (inferences)*) For all sets of crowdworking 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 (where the disclosure sets are defined in Section 3).

Second, we extend the possible behaviors to malicious behaviors aiming at jeopardizing regulations and show that they are systematically detected by SEPAR (Lemma 1 focuses on enforceable regulations and Lemma 2 on verifiable regulations). This prevents covert adversaries to perform malicious actions, limiting them to inferences.

Lemma 1. (*Detection of malicious behaviors (enforceable regulations)*) A crowdworking 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 enforceable regulation.

Lemma 2. (*Detection of malicious behaviors (verifiable regulations)*) A participant can produce a proof about a crowdworking 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 completed successfully.

Since Theorem 1 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 2).

Theorem 2. (*Privacy (inferences and malicious behaviors)*) For all sets of crowdworking 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 (where the disclosure sets are defined in Section 3).

5 Coping with Distribution

Section 4 provides an abstract design for a token-based system that enforces global constraints on transaction executions. In this section, we develop *SEPAR*, a multi-platform crowdworking system that supports the execution of transactions on multiple globally distributed platforms that do not necessarily trust each other. Furthermore, to provide fault tolerance, each platform is implemented on a set of nodes (i.e., replicas) that store copies of the platform's ledger. SEPAR uses a *permissioned blockchain* as its underlying infrastructure where the ledger is implemented as blockchain ledger.

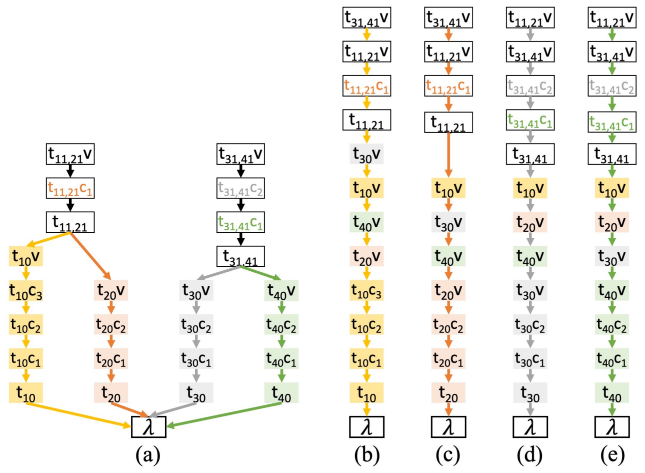


Figure 2: (a): The ledger of SEPAR with 4 platforms, (b), (c), (d), and (e): The views of the ledger from different platforms

The unique features of blockchain such as transparency, provenance, fault tolerance, and authenticity are used by many systems to deploy a wide range of distributed applications in a permissioned settings. In particular and for a crowdworking application, the *transparency* of blockchains can be used to check integrity constraints, *provenance* enables the system to trace how data is transformed, *fault tolerance* helps to enhance reliability and availability, and finally, *authenticity* guarantees that signatures and transactions are valid.

5.1 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 order of transactions in the blockchain ledger is captured by *chaining* transaction blocks together, i.e., each transaction block includes the 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, the ledger (similar to [1, 2]) 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.

Fig. 2(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 unique initialization (*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. 2(a), t_{10}, t_{20}, t_{30}, and t_{40} are internal submission transactions of different platforms that can be appended to the ledger in parallel. As shown, t_{10} requires 3 contributions (thus 3 claim transactions $t_{10}c_1$, $t_{10}c_2$, and $t_{10}c_3$) 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 the corresponding 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, $t_{11,21}$ and $t_{31,41}$ require one and two contributions respectively. 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).

This global directed acyclic graph blockchain ledger includes all transactions of internal as well as cross-platform tasks initiated by all platforms. However, to ensure data privacy, each platform should only access the transactions in which the platform is involved. As a result, in SEPAR, the entire blockchain ledger is *not maintained* by any specific 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 involving the platform, and (3) verification transactions of all tasks. Note that verification transactions are replicated on every platform to enable all platforms checking the satisfaction of global regulations. The global directed acyclic graph ledger is indeed the union of all these physical views.

Fig. 2(b)-(e) show the views of the ledger for platforms p_1, p_2, p_3, and p_4 respectively. As shown, 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 and verification transactions of all tasks. Note that, since there is no data dependency between the verification transactions of the tasks that involve a particular platform and 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.

5.2 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 the crash or Byzantine failure model. In the crash failure model, nodes may fail by stopping, and may restart, however, in the Byzantine failure model, faulty nodes may exhibit 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, e.g., Paxos [21], 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, e.g., PBFT [10], $3f+1$ nodes are usually needed to provide safety in the presence of f malicious nodes.

Completion of a crowdworking task, as discussed earlier, requires a single submission transaction, one or more claim transactions, 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.

5.2.1 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, a crash, e.g., Paxos [21], a Byzantine, e.g., PBFT [10], or a hybrid, e.g., SeeMoRe [3] fault-tolerant protocol can be used.

5.2.2 Cross-Platform and Global Consensus.

Both cross-platform consensus and global consensus require collaboration between multiple platforms. Since platforms do not trust each other and the primary node that initiates the transaction might behave maliciously, SEPAR uses asynchronous *Byzantine* fault-tolerant protocols in both cross-platform and global consensus. cross-platform and global consensus, however, are different in two aspects. First, in cross-platform consensus only the involved platforms participate, whereas global consensus is established among all platforms, and second, at the platform level, while cross-platform consensus requires agreement from every *involved* platform, in global consensus agreement from two-thirds of all platforms is sufficient. Cross-platform consensus requires agreement from every involved platform to ensure data consistency due to the possible data dependency between the cross-platform transaction and other transactions of an involved platform. Note that if an involved platform (as a set of nodes) behaves maliciously by not sending agreement for a cross-platform transaction initiating by another platform, e.g., a claim transaction, its malicious behavior can be detected and penalties imposed. In global consensus, however, the goal is only to check the correctness of the transaction. To provide safety for global consensus at the platform level, we assume that at most $\lfloor \frac{|P|-1}{3} \rfloor$ platforms might behave maliciously. As a result, to commit a transaction, by a similar argument as in PBFT [10], at least two-thirds ($\lfloor \frac{2|P|}{3} \rfloor + 1$) of the platforms must agree on the order of the transaction.

Cross-Platform Consensus. Processing Submission and claim transactions of a cross-platform task requires cross-platform consensus among *all* involved platforms where due to the untrustworthiness of platforms a Byzantine fault-tolerant protocol is used. Since the number of nodes within each platform depends on the failure model of nodes of a platform (i.e. $2f + 1$ crash-only or $3f + 1$ Byzantine nodes), the required number of matching replies from each platform, i.e., the quorum size, to ensure the safety of protocol is different for different platforms. We define *local-majority* 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 established consensus on cross-platform transactions in four phases: (1) prepare, (2) propose, (3) accept, and (4) commit, and then (5) appends transaction to the ledger.

(1) Prepare Phase. Upon receiving a cross-platform (submission or claim) transaction m, the (pre-elected) node of the (recipient) platform p_i (called the *initiator primary*) initiates the consensus protocol by assigning a sequence number h_i to the transaction and multicasting a signed prepare message μ to the primary node of all involved platforms. The prepare message includes the received transaction m (either submission or claim), its digest $d{=}D\,(m)$, and the sequence number h_i. The sequence number represents the correct order of the transaction block in the initiator platform. If the transaction is a claim transaction, the initiator primary includes the hash of the corresponding submission transaction as well.

(2) Propose Phase. Once the primary node of some platform p_j receives a prepare message μ for some transaction m from the initiator primary, it first validates the message. If the node 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 in p_j as well as some other cluster p_k, 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 overlapping clusters (Consistency), e.g., m and m' are committed in the same order on both p_j and p_k. SEPAR addresses deadlock situation, i.e., where overlapping clusters receive prepare messages in different order, in the same way as SharPer [2]). If the primary is not waiting for an uncommitted transaction, it assigns sequence number h_j (that represents the order of m in platform p_j) to the message and multicasts a *signed* propose message to the nodes of *its* platform including both sequence numbers h_i and h_j, digest d and piggybacked prepare message μ. The initiator primary node, similarly, multicasts a signed propose message (including h_i, d, and piggybacked μ) to the nodes of its platform.

(3) Accept Phase. Once a node of an involved platform p_j receives a valid propose message from the primary node of its cluster which has no overlap with any uncommitted cross-platform requests, the node multicasts a signed accept message including both sequence numbers h_i and h_j, and digest d to *every* node of *all* involved platforms. The primary nodes of all involved platforms also multicast accept messages (with the same structure) to *every* node of *all* involved platforms. Note that if a platform behaves maliciously by not sending accept messages to other involved platforms, the initiator platform can report the malicious behavior of the platform by sending an ALERT message (similar to Section **??**) to the registration authority resulting in imposing penalties.

(4) Commit Phase. Upon receiving 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 match the propose message which was sent by primary, every node multicasts a signed commit message including all valid sequence numbers, e.g., $h_i, h_j, ..., h_k$, to all nodes of every involved platforms. The prepare, propose and accept phases of the algorithm guarantee that non-faulty nodes agree on a total order for the transactions.

(5) Append to Ledger. Finally, each node waits for valid matching commit messages from a local majority of *every* involved platform that match its commit message before committing the transaction. 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 messages to its copy of the ledger. Note that since commit messages include the digest of the corresponding transactions, appending valid signed commit messages to the blockchain ledger in addition to the transactions, provides the same level of *immutability* guarantee as including the cryptographic hash of the previous transaction in the transaction block, i.e., any attempt to alter the block data can easily be detected.

In terms of message complexity, prepare phase consists of $|P|$ messages, propose phase needs n messages, and accept and commit phases, each requires n^2 messages where n is the number of nodes.

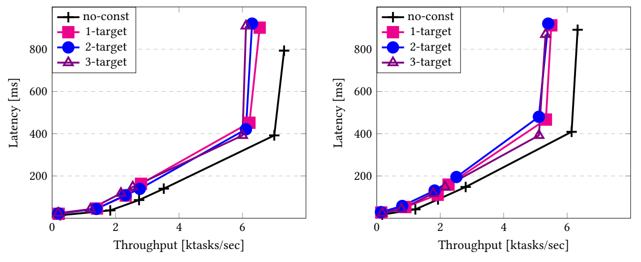
In addition to the normal case operation, SEPAR has to deal with two other scenarios. First, when the primary node fails. Second, when nodes have not received a quorum of *matching* accept messages from the local-majority of every involved platform due to conflicting accept messages. Indeed, the primary 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 primary of their own platform (because of the asynchronous network). We use techniques similar to SharPer [2] to address these two situations. Due to space limitation, the detailed explanation of the techniques is omitted and are provided in the extended version of the paper [due to double-blind policy, we cannot provide (reference) the full version of the paper].

Global Consensus. The verification transactions include group signatures and all tokens that are consumed by 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, similar to cross-platform consensus, is run among all nodes of every platform where in each phase, the protocol needs agreement from the *local majority* of *two-thirds* of the platforms. The send an agreement, nodes validate the transaction including the group signatures and consumed tokens.

The correctness of both cross-platform and global consensus protocols is proven in the extended version of the paper [due to double-blind policy, we cannot reference the extended version].

6 Experimental Evaluations

In this section, we conduct several experiments to evaluate the scalability and privacy of SEPAR. We consider a complex heavily loaded setting with 4 platforms (except for the last experiment), 20000 requesters and 20000 workers where both requesters and workers are randomly registered to one or more platforms. Once a task is submitted to a platform, the platform randomly assigns the task to one or more (depending on the required number of contribution) idle workers (to avoid any delay). The experiments consists of two main parts. In the first part (Sections 6.1 and 6.2), the privacy aspect of SEPAR (i.e., tokens and constraints) is evaluated, whereas in the second part (Sections 6.3 and 6.4), the scalability of system is evaluated. For the purpose of this 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 we use the protocol proposed in [9]. 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.



(a) Crash-Only Nodes (b) Byzantine Nodes

Figure 3: Varying the Type of Constraints

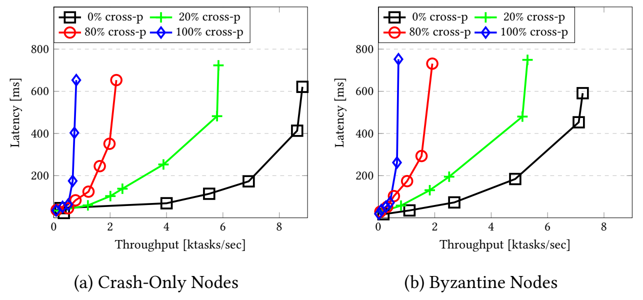
6.1 Token Generation

In the first set of experiments, we measure the performance of token generation (performed by RA) in SEPAR for different types of constraints. We consider different types of constraints, i.e., single-target (e.g., $((w, *, *), \theta)$), two-target (e.g., $((*, p, r), \theta)$), and three-target (e.g., $((w, p, r), \theta)$) constraints. Our experiments shows that SEPAR is able to generate tokens in linear time. SEPAR generates each token in $0.07ms$, hence, generating 1 million tokens in 76 seconds. This clearly demonstrates the scalability of token generation specially since token generation is executed periodically, e.g., every week or every month. Note that, we use a single machine to generate tokens, however, tokens related to different constraints can be generated in parallel. Hence, the throughput of SEPAR can linearly increase by running the token generation routine on multiple machines. Furthermore, types of constraints (i.e., the number of targets) does not affect the performance and the throughput and latency of token generation is constant in terms of the number of targets. It should, however, be noted that a constraint with more targets requires more tokens to be generated, e.g., $((w, *, *), \theta)$ requires $|w| * \theta$ tokens, whereas $((w, p, r), \theta)$ requires $|w| * |p| * |r| * \theta$ tokens to be generated.

6.2 Type of Constraints

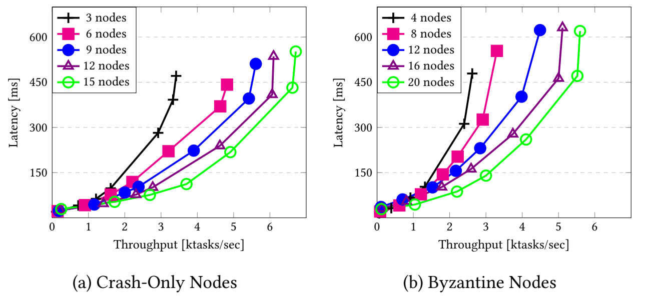
In the second set of experiments, we measure the overhead of privacy-preserving techniques, e.g., group signatures and tokens, used in SEPAR. We consider the basic scenario with no constraint (i.e., no need to exchange and validate tokens and signatures) and compare it with three different scenarios where each task has to satisfy a single target, a two-target, and a three-target constraint. The system consists of four platforms and the workload includes 90% intra- and 10% cross-platform tasks (the typical settings in partitioned databases [30, 31]) where two (randomly chosen) platforms are involved in submission and claim transactions of cross-platform tasks (note that all platforms are still involved in the verification transaction of each task). We also assume that completion of each task requires a single contribution (and obviously a submission and a verification transaction).

When nodes follow the crash failure model and the system has no constraints, as can be seen in Figure 3(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



(a) Crash-Only Nodes (b) Byzantine Nodes

Figure 4: Varying Number of Cross-Platform Tasks



(a) Crash-Only Nodes (b) Byzantine Nodes

Figure 5: Varying the Number of Platforms

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. This results demonstrates that all privacy-preserving techniques that are used in SEPAR results in only 11% and 15% overhead in terms of the throughput and latency respectively). Moreover, the type of constraints does not significantly affect the performance of SEPAR. This is expected, because more targets results in only increasing the number of (parallel) tokens and signature exchanges while the number of communication phases is not affected.

Similarly, in the presence of Byzantine nodes and as shown in Figure 3(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 single-target constraints. As before, the type of constraints does not affect the performance.

It should be noted that by increasing the number of constraints, SEPAR still demonstrates similar behavior as shown in this experiment. Indeed, adding more constraints, while it results in adding more tokens and possibly more participants and signatures, it does not affect the consensus protocols and other communication phases.

6.3 Cross-Platform Tasks

In the next 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 (4) 100% cross-platform tasks. As before, completion of each task requires a single contribution and two (randomly chosen) platforms are involved in each cross-platform task. 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 4(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 4(b), SEPAR demonstrates the similar behavior as the crash-only case.

6.4 The Number of Platforms

In the last set of experiments, we measure the scalability of SEPAR in crowdworking environments with different number of platforms (1 to 5 platforms) for both crash-only and Byzantine nodes (assuming $f = 1$ in each platform). The networks, thus, include 3, 6, 9, 12, and 15 crash-only nodes or 4, 8, 12, 16 and 20 Byzantine nodes. Each task has to satisfy 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 local. As shown in Figure 5(a), in the presence of crash-only nodes, the performance of the system is improved 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 increases 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.

7 Related Work

Enhancing privacy in the context of crowdworking has been addressed by several recent studies with various kinds of guarantees, from differential privacy [32, 33] to cryptography [22–24], mostly focusing on spatial crowdsourcing and the use of geolocation to perform assignments. In ZebraLancer [25] and ZKCrowd [35], blockchain is also used to add transparency guarantees on top of privacy. However, all these works consider a single-platform context. Other works propose to manage multiple platforms in crowdsourcing. For instance, Fluid [16] proposes to share workers' profiles between multiple platforms, and Wang et al. [34] provides an interesting insight on federated recommendation systems for workers and the balance of surplus of tasks or workers in a cross-platform context. However, none of them provides tools to manage external regulations: to the best of our knowledge, SEPAR is the first to support a multi-platform crowdworking context, with external constraints, transparency, and privacy expectations at the same time.

In the context of permissioned blockchains, Hyperledger Fabric [4] ensures data confidentiality using Private Data Collections[20]. Private Data Collections manage confidential data that two or more entities want to keep private from others. Quorum [12] supports public and private transactions and ensures the confidentiality of private transactions using the Zero-knowledge proof technique. Quorum, however, orders all public and as private transactions using a single consensus protocol resulting in low throughput. SEPAR is inspired by various permissioned blockchain systems, and more specifically by SharPer [2], a permissioned blockchain system that uses sharding to improve scalability. Unlike SEPAR, SharPer is designed for environments with a single application enterprise, which would correspond to the ledger of a single platform in SEPAR. Furthermore, SharPer supports simple transactions that are all processed in the same way, while SEPAR has different types (i.e., submission, claim, and verification) and depending on its type, the transactions will be processed in different ways, e.g., verification transactions require coordination among all platforms. In SharPer, the infrastructure consists of clusters of nodes where the single ledger is partitioned into shards that are assigned to different clusters. SEPAR, on the other hand, includes multiple enterprises (i.e., platforms) that might not trust each other. As a result, while each shard in SharPer is similar to a platform in SEPAR, any cross-platform communication in SEPAR requires a Byzantine fault-tolerant protocol that tolerates malicious behaviour at the platform level. This is in contrast to SharPer where the type of cross-platform consensus depends on the failure model of nodes (and not the application, which is always trusted). SEPAR needs to support, in addition to local and cross-platform consensus, a global consensus protocol that establishes agreement among all (possibly untrusted) platforms on the order of verification transactions. Finally, while in SharPer all nodes are either crash-only or Byzantine, in SEPAR, platforms are run by different enterprises and hence can have different failure models.

Providing anonymity as well as untraceability has been addressed by ZCash [17] which is restricted to the management of crypto-currency issues. Hawk [19] and Raziel [29] manage wider issues, and include general smart contracts. However, these solutions do not incorporate infrastructures with multiple platforms, nor implement constraints (let alone anonymized ones). Finally, Solidus [11] proposes to privately manage a multi-platform banking system, with individual banks managing their own clients while allowing cross-platform transactions. While Solidus may be sufficient for banking systems, it does not consider users that subscribe to multiple platforms, nor envisions global profiles or constraints.

8 Conclusion

In this paper, we present an overall vision for multi-platform crowdworking environments consisting of three main dimensions: regulations, security and architecture. We then introduce SEPAR, (to the best of our knowledge) the first to address the problem of enforcing global regulations over multi-platform crowdworking environments. SEPAR enables official institutions to express global regulations in simple and unambiguous terms, guarantees the satisfaction of global regulations by construction, and allows participants to prove to external entities their involvement in crowdworking tasks, all in a privacy-preserving manner. SEPAR also uses transparent blockchain ledgers shared across multiple platforms and enables collaboration among platforms through a suite of distributed consensus protocols. We prove the privacy requirements of SEPAR and conduct an extensive experimental evaluation to measure the performance and scalability of SEPAR.

[20] www.hyperledger.org/blog/2018/10/23/private-data-collections-a-high-level-overview

References

[1] M. J. Amiri, D. Agrawal, and A. El Abbadi. 2019. CAPER: a cross-application permissioned blockchain. *VLDB* 12, 11 (2019), 1385–1398.
[2] M. J. Amiri, D. Agrawal, and A. El Abbadi. 2019. SharPer: Sharding Permissioned Blockchains Over Network Clusters. *arXiv preprint arXiv:1910.00765* (2019).
[3] M. J. Amiri, S. Maiyya, D. Agrawal, and A. El Abbadi. 2020. SeeMoRe: A Fault-Tolerant Protocol for Hybrid Cloud Environments. (2020), 1345–1356.
[4] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, et al. 2018. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *EuroSys*. ACM.
[5] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. 2000. A practical and provably secure coalition-resistant group signature scheme. In *CRYPTO*. Springer, 255–270.
[6] Y. Aumann and Y. Lindell. 2007. Security against covert adversaries: Efficient protocols for realistic adversaries. In *TCC*. Springer, 137–156.
[7] J. Berg, M. Furrer, E. Harmon, U. Rani, and M S. Silberman. 2018. *Digital labour platforms and the future of work : Towards decent work in the online world.* Technical Report ISBN 978-92-2-031025-0. International Labour Organization.
[8] P. Bourhis, G. Demartini, S. Elbassuoni, E. Hoareau, and H. R. Rao. 2019. Ethical Challenges in the Future of Work. *IEEE Data Eng. Bull.* 42, 4 (2019), 55–64.
[9] J. Camenisch and J. Groth. 2004. Group signatures: Better efficiency and new theoretical aspects. In *SCN*. Springer, 120–133.
[10] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186.
[11] E. Cecchetti, F. Zhang, Y. Ji, A. Kosba, A. Juels, and E. Shi. 2017. Solidus: Confidential distributed ledger transactions via PVORM. In *ACM CCS*. 701–717.
[12] JP Morgan Chase. 2016. Quorum white paper.
[13] D. Chaum and E. Van Heyst. 1991. Group signatures. In *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 257–265.
[14] J. E. Cohen. 2017. Law for the platform economy. *UCDL Rev.* 51 (2017), 133.
[15] D. Gross-Amblard, A. Morishima, S. Thirumuruganathan, M. Tommasi, and K. Yoshida. 2019. Platform Design for Crowdsourcing and Future of Work. *IEEE Data Eng. Bull.* 42, 4 (2019), 35–45.
[16] S. Han, Z. Xu, Y. Zeng, and L. Chen. 2019. Fluid: A blockchain based framework for crowdsourcing. In *SIGMOD*. 1921–1924.
[17] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox. 2016. Zcash protocol specification. *GitHub: San Francisco, CA, USA* (2016).
[18] M. Kenney and J. Zysman. 2016. The rise of the platform economy. *Issues in science and technology* 32, 3 (2016), 61.
[19] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. 2016. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *S&P*. IEEE, 839–858.
[20] Leslie Lamport. 1978. The implementation of reliable distributed multiprocess systems. *Computer Networks (1976)* 2, 2 (1978), 95–114.
[21] L. Lamport. 2001. Paxos made simple. *ACM Sigact News* 32, 4 (2001), 18–25.
[22] A. Liu, Z. Li, G. Liu, K. Zheng, M. Zhang, Q. Li, and X. Zhang. 2017. Privacy-preserving task assignment in spatial crowdsourcing. (2017).
[23] A. Liu, W. Wang, S. Shang, Q. Li, and X. Zhang. 2018. Efficient task assignment in spatial crowdsourcing with worker and task privacy protection. *GeoInformatica* 22, 2 (2018), 335–362.
[24] B. Liu, L. Chen, X. Zhu, Y. Zhang, C. Zhang, and W. Qiu. 2017. Protecting location privacy in spatial crowdsourcing using encrypted data. *EDBT* (2017).
[25] Y. Lu, Q. Tang, and G. Wang. 2018. Zebralancer: Private and anonymous crowdsourcing system atop open blockchain. In *ICDCS*. IEEE, 853–865.
[26] Conseil national du numérique. 2020. Travail à l'ère des plateformes. Mise à jour requise. https://cnnumerique.fr/publication-du-rapport-travail-lere-des-plateformes-mise-jour-requise-en-presence-de-cedric-o (in French).
[27] Global Commission on the Future of Work. 2019. *Work for a brighter future.* Technical Report ISBN 978-92-2-132796-7. International Labour Organization.
[28] FoW participants. 2019. Imagine all the People and AI in the Future of Work. ACM SIGMOD blog post.
[29] D. C. Sánchez. 2018. Raziel: Private and verifiable smart contracts on blockchains. *arXiv preprint arXiv:1807.09484* (2018).
[30] R. Taft, E. Mansour, M. Serafini, J. Duggan, A. J Elmore, A. Aboulnaga, A. Pavlo, and M. Stonebraker. 2014. E-store: Fine-grained elastic partitioning for distributed transaction processing systems. *VLDB* 8, 3 (2014), 245–256.
[31] A. Thomson, T. Diamond, S. Weng, K. Ren, P. Shao, and D. J. Abadi. 2012. Calvin: fast distributed transactions for partitioned database systems. In *SIGMOD*. 1–12.

[32] H. To, G. Ghinita, L. Fan, and C. Shahabi. 2016. Differentially private location protection for worker datasets in spatial crowdsourcing. *IEEE Transactions on Mobile Computing* 16, 4 (2016), 934–949.
[33] H. To, C. Shahabi, and L. Xiong. 2018. Privacy-preserving online task assignment in spatial crowdsourcing with untrusted server. In *ICDE*. IEEE, 833–844.
[34] Y. Wang, T. Song, Q. Tao, Y. Zeng, Z. Zhou, Y. Xu, Y. Tong, and L. Chen. 2019. Interaction Management in Crowdsourcing. *Data Engineering* (2019), 23.
[35] S. Zhu, Z. Cai, H. Hu, Y. Li, and W. Li. 2019. zkCrowd: a hybrid blockchain-based crowdsourcing platform. *Transactions on Industrial Informatics* (2019).